The Dissertation Committee for Jae Yong Chung
certifies that this is the approved version of the following dissertation:

Refactoring-based Statistical Timing Analysis and Its Applications to Robust Design and Test Synthesis

Committee:

Jacob A. Abraham, Supervisor

Nur A. Touba

Adnan Aziz

Michael Orshansky

Yaping Zhan

Refactoring-based Statistical Timing Analysis and Its Applications to Robust Design and Test Synthesis

by

Jae Yong Chung, B.S.;M.S.E.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

May 2011

Dedicated to my father, Sukku Chung,

and my mother, Myungsuk Kim.

Acknowledgments

I would like to express my deepest gratitude to my advisor, Dr. Jacob A. Abraham, for his excellent guidance, continuous support, patience, and providing me with an excellent atmosphere for doing research which allowed me to work on fundamental research problems. Without his guidance this dissertation would not have been completed. I would like to thank Dr. Nur Touba, Dr. Adnan Aziz, Dr. Michael Orshansky and Dr. Yaping Zhan for their time and insightful comments while being members of my dissertation committee.

My gratitude goes to my friends in the Computer Engineering Research Center, Joonsung Park, Jae-Wook Lee, Jihwan Chun, Kihyuk Han, Eun Jung Jang, Hyun Jin Kim, Junyoung Park, Joonsoo Kim, Ashutosh Chakaraborty, Minsik Cho, Hongjung Shin, Jinkyu Lee, Wooyoung Jang, Yongchan Ban, Kun Yuan, Rajeshwary Tayade, Sriram Sambamurthy, Sankar Gurumurthy, Mahesh Prabhu. I would like to appreciate ECE friends, Donghyuk Shin, Ick-Jae Yoon, Jungho Jo, Joon-Sung Yang, Yonghyun Kim, Bong Wan Jun, Wonsoo Kim, Taesoo Jun, Jae Hong Min, Minsoo Rhu, Jinsuk Chung. Special thanks to my roommate, Seyoung Kim, who has been with me for entire years of my study, both in times of deep despair and in times of joy. Also, I would like to acknowledge Melissa Campos and Debi Prather for administrative support.

I am very grateful to my colleagues at IBM T.J. Watson Research Center, Jinjun Xiong and Vladimir Zolotov. They are world-class experts in VLSI CAD area, and I was fortunate to work together with them for state-of-the-art timing analysis technology.

My two internships at Strategic CAD Lab.(SCL), Intel and IBM T.J. Watson Research Center provided me excellent opportunities to learn cutting-edge technologies and industrial problems. I would like to thank Suriyaprakash Natarajan and Eli Chiprout from Intel, and Tom Fox, Jun Sawada, Daniel A. Prener and Bill Reohr from IBM. Also, I would like to express my gratitude to Samsung Electronics and Intel for their financial support.

Finally, and most importantly, I am indebted to the love and support that I have received from my parents. In special, I am thankful to Hyunyoung Choi for her love, encouragement, sacrifice, and support.

Refactoring-based Statistical Timing Analysis and Its Applications to Robust Design and Test Synthesis

Publication No. _____

Jae Yong Chung, Ph.D.
The University of Texas at Austin, 2011

Supervisor: Jacob A. Abraham

Technology scaling in the nanometer era comes with a significant amount of process variation, leading to lower yield and new types of defective parts. These challenges necessitate robust design to ensure adequate yield, and smarter testing to screen out bad chips. Statistical static timing analysis (SSTA) enables this but suffers from crude approximation algorithms.

This dissertation first studies the underlying theories of timing graphs and proposes two fundamental techniques enhancing the core statistical timing algorithms. We first propose the refactoring technique to capture topological correlation. Static timing analysis is based on levelized breadth-first traversal, which is a fundamental graph traversal technique and has been used for static timing analysis over the past decades. We show that there are numerous alternatives to the traversal because of an algebraic property, the distributivity of addition over maximum. This new interpretation extends the degrees of

freedom of static timing analysis, which is exploited to improve the accuracy of SSTA. We also propose a novel operator for computing joint probabilities in SSTA. In many SSTA applications, this is very common but is done using the max operator which results in much error due to the linear approximation. The new operator provides significantly higher accuracy at a small cost of run time.

Second, based on the two fundamental studies, this dissertation develops three applications. We propose a criticality computation method that is essential to robust design and test synthesis; The proposed method, combined with the two fundamental techniques, achieves drastic accuracy improvement over the state-of-the-art method, demonstrating the benefits in practical applications. We formulate the statistical path selection problem for at-speed test as a gambling problem and present an elegant solution based on the Kelly criterion. To circumvent the coverage loss issue in statistical path selection, we propose a testability driven approach, making it a practical solution for coping with parametric defects.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The shrinking feature sizes of integrated circuits cause chips to operate faster and consume less power, which adds value to the chips. The added value makes us willing to pay the price of next technology scaling, and this positive cycle has been fueled the silicon industry over the past decades. However, we are facing the breakdown of the cycle. One reason is that it is becoming difficult to control device parameters precisely. In other words, the variability in process parameters is increasing. In this situation, the traditional margining to deal with the variability depreciates the chips, stopping the positive cycle. Significant efforts have been devoted to coping with the variability issues in the nanometer IC technologies, which results in statistical static timing analysis (SSTA). The initial efforts to deploy the statistical timing methodology such as modeling, measurement, and building timing library may be large, but the benefits of using SSTA come at almost every stage in IC development. It provides much room for improvement in various CAD algorithms at many different levels, and the efforts to harvest the fruits of statistical timing are ongoing in industry and academies.

Current block-based SSTA is efficient and accurate enough to be used

for timing sign-off and can produce the probability distribution function (PDF) of the circuit slack pretty accurately. However, developing applications on top of SSTA reveals that large inaccuracy is hidden behind the PDF. In particular, there are two well-known sources of inaccuracy. If two arrival times come through a common path, they are correlated due to the topology. This is called the *topological correlation* which is ignored in most SSTA tools. The maximum of two delays become a non-linear function to the process parameters, but approximate it by a linear function which causes some error. This error is called the *linear approximation error* of the max operator. These two sources of inaccuracy hinder the development of SSTA applications in design automation and VLSI testing, while the significance of such applications is growing as variability increases.

1.1 Robust Design Applications

Using SSTA, designers can validate before tape-out that chips will be produced at a high yield even under various sources of variation. If a design is too sensitive to some sources of variation, designers will fix the design in order to reduce the sensitivities, making it robust. Thus, SSTA should be able not only to predict the yield accurately, but also to assist them in enhancing the design to improve the productivity of designers. Also, since a large part of today's designs comes from CAD tools, the sensitivities should be translated into a form that can be used easily in high-level CAD algorithms. The sensitivity values themselves are difficult to deal with by designers or the algorithms, so

we usually compute certain probability values from the sensitivities. The timing criticality of a path or an edge is a probability value widely used for this purpose, and it allows us to locate paths or edges that should be optimized to enhance the design. Designers can identify critical paths or edges to be targeted, and we can develop statistical design automation algorithms using the criticality. For example, a statistical discrete gate sizing algorithm sizes up the gates with high edge criticality and sizes down those with low edge criticality iteratively. However, current efficient computational methods add a significant amount of error in the process of the translation from the sensitivities to the probability values, and the error leads to sub-optimal robust designs. Thus, we require computational methods with higher accuracy for computing the probabilities.

1.2 VLSI Testing Applications

Due to the ever increasing variation, yield loss caused by parametric faults is greater than that by catastrophic faults. Catastrophic faults have been of concern for a very long time, because they are a main yield loss mechanism in previous technologies. To capture their behaviors in the functional domain, various fault models such as the stuck-at fault model and the transition fault model have been proposed, and many effective testing methodologies have been developed using these models. Therefore, defective parts have been screened without significant increase of the test cost. However, effective testing methodologies to deal with parametric faults do not yet exist.

In the models for catastrophic faults, fault coverage has been widely used as a test quality metric, by which tests are evaluated. If test patterns are evaluated as insufficient, we can add more test vectors. Test vectors come from various sources. Sometimes they are written manually. They are often generated randomly or by an ATPG (automatic test pattern generation) tool. Also, they can be part of existing applications. From these sources, we select more test vectors and add them to the test flow. However, since this will increase the test time and test cost, we are encouraged to improve test techniques and develop better test methodologies. ATPG algorithms can be elaborated, or better test compaction techniques can be introduced. Random test vector generators, no matter where they exist (e.g., in ATPG tools or on chip for built-in self test) can be biased to increase fault coverage. If all these efforts are not effective, we may try design-for-test (DFT). For example, by inserting scan chains, the complex sequential ATPG problem can be changed into a simple combinational ATPG problem which allows us to achieve high test coverage at small cost. Through these efforts, test patterns become sufficient to meet a desired test quality goal. If test patterns are evaluated as exceeding the goal, we can remove some test vectors from the test flow, which will decrease the test time and test cost. We may be able to remove some DFT circuits to save area and power. This testing eco-system is enabled by a simple and effective test metric. For catastrophic faults, fault coverage delivers the goods as the test metric, since it is easy to calculate and well correlated to DPPM (defective parts per million), an ultimate test quality measure used in

testing.

However, fault coverage for parametric faults is not an effective test metric. Conventionally, the path delay fault model has dealt with parametric delay faults. The fault coverage of the path delay fault model is the percentage of paths to be tested over the number of total paths. The fault coverage in the path delay fault model is not well correlated with DPPM, because the fault probabilities of paths are very different and the variations of path delays have complex statistical correlation. Due to the absence of an effective test metric, the testing eco-system does not work properly. Not only tests do not have been evaluated correctly, but also the need for better test techniques and methodologies has been hidden.

We believe SSTA can play a key role in the testing eco-system for parametric faults for several reasons. First, it is efficient. It is a linear time method with respect to the circuit size. Second, it can calculate DPPM directly, so the correlation between the test metric and the DPPM need not be worried. Third, a lot of effort has already been made to develop SSTA methodology, and the algorithms and modeling techniques for SSTA are already mature. Therefore, we can develop testing methodologies on top of them.

1.3 Parameterized Statistical Timing

Most recent SSTA algorithms [60, 10, 71] employ a parameterized delay model to efficiently capture correlation between delays. Under the linear

parameterized delay model, a (gate or wire) delay d is represented as

$$d = e_0 + \sum_{i=1}^{n} e_i \Delta X_i + e_{n+1} \Delta R_d \tag{1.1}$$

where $e_0 = E[d]$, $e_i = \partial d / \partial \Delta X_i$, $e_{n+1} = \partial d / \partial \Delta R_d$, ΔX_i is a normalized random variable representing a global source of variation, and ΔR_d is a normalized random variable representing the independent random variation of d. The independent random variation is a source of variation that affects the delay d only (e.g. random dopant fluctuation of V_{th}). The global source of variation is a source of variation that affects many delays.

The sum of two timing quantities represented in the form of (1.1) can be easily represented in the same form again. However, the maximum of two timing quantities of the form (1.1) does not fit in the same form, but we represent it approximately in the form using moment-matching type algorithms [10, 71]. Therefore, all timing quantities such as delay, arrival times, required arrival times, slacks are represented in the same form, and we call the form the *canonical form* [71]. Given two timing quantities A and B represented in the canonical form, we can easily obtain

$$P(A \geq B) = \Phi(\alpha) \tag{1.2}$$

where

$$\begin{aligned} \alpha &= (E[A] - E[B])/a, \\ a &= \sqrt{var[A] + var[B] - 2cov[A, B]}, \\ \Phi(x) &= \int_{-\infty}^{x} \varphi(t)dt \text{ and } \varphi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}. \end{aligned} \tag{1.3}$$

This is shown in [17] and this probability is called the *tightness probability* of A over B.

1.4 Organization of the Dissertation

In Chapter 2 we present a technique for capturing topological correlation in arbitrary block-based statistical static timing analysis (SSTA). We interpret a timing graph as an algebraic expression made up of addition and maximum operators. We define the division operation on the expression and propose algorithms that modify factors in the expression without expansion. As a result, the algorithms produce an expression to derive the latest arrival time with better accuracy in SSTA. Existing techniques handling reconvergent fanouts usually use dependency lists, requiring quadratic space complexity. Instead, the proposed technique has linear space complexity by using a new directed acyclic graph search algorithm. Our results show that it outperforms an existing technique in speed and memory usage with comparable accuracy. More importantly, the proposed technique is not limited to SSTA and is potentially applicable to various issues due to reconvergent paths in timing-related CAD algorithms.

In Chapter 3, we present a method to compute criticality probabilities of paths in parameterized statistical static timing analysis (SSTA). We partition the set of all the paths into several groups and formulate the path criticality into a joint probability of inequalities. Before evaluating the joint probability directly, we simplify the inequalities through algebraic elimina-

tion, handling topological correlation. Our proposed method uses conditional probabilities to obtain the joint probability, and statistics of random variables representing process parameters are changed because of given conditions. To calculate the conditional statistics of the random variables, we derive analytic formulas by extending Clark's work. This allows us to obtain the conditional probability density function of a path delay, given the path is critical, as well as to compute criticality probabilities of paths. Our experimental results show that the proposed method provides 4.2X better accuracy on average in comparison to the state-of-art method.

In Chapter 4, we present a new path selection algorithm for delay fault testing in a statistical timing framework. Existing algorithms which consider correlation between path delays use an iterative process for each path or defect and require a Monte Carlo simulation for each iteration to calculate the conditional fault probability. The proposed algorithm does not require the iteration process and selects a requested number of paths simultaneously once it performs a statistical timing analysis at the beginning. If selection of k paths is required in a set of paths, it partitions the set into two path sets and determines how many paths should be selected in each path set out of the k paths. It recursively continues this process and ends up with k paths. The partitioning is easily performed during the recursive traversal of a circuit, which produces an implicit path tree, where paths are already grouped based on their prefix or suffix. Experimental results show the proposed algorithm can effectively use the correlation to generate high quality path sets. In

the face of large-scale process variations, statistical timing methodology has advanced significantly over the last few years, and statistical path selection takes advantage of it in at-speed testing. In deterministic path selection, the separation of path selection and test generation is known to require time consuming iteration between the two processes. In Chapter 5, we show that this in statistical path selection is not only the case, but also the quality of results can be severely degraded even after the iteration. To deal with this issue, we consider testability in the first place by integrating a SAT solver, and this necessitates a new statistical path selection method. We integrate the SAT solver in a novel way that leverages the conflict analysis of modern SAT solvers which provides more than 4X speed-up, without special optimizations of the SAT solver for this particular application. Our proposed method is based on a generalized path criticality metric which properties allow efficient pruning. Our experimental results show that the proposed method achieves 47% better quality of results on average, and up to 361x speedup compared to statistical path selection followed by test generation.

Chapter 2

A Hierarchy of Subgraphs Underlying a Timing Graph and Its Use in Capturing Topological Correlation

2.1 Introduction

The increasing variability in the nanometer regime, combined with shrinking technology parameters, has posed new challenges in static timing analysis. Circuit timing is subject to many factors including process parameters, voltage and temperature. These are given as parameters in the timing analysis flow. Some of them are provided as fixed constants and others are provided as uncertain variables with ranges or probabilistic distributions. The increasing variability requires timing analysis to take the parameters that were considered as fixed constants in the previous technologies as uncertain variables. It is too pessimistic to assume the worst case for the increased number of uncertain parameters, and the traditional corner-based approach does not scale well with the increased number of parameters. Thus a research direction has been to look for an alternative timing analysis method.

Block-based statistical timing analysis is a popular method. In the early stage of the development of the approach, each gate and wire delay was

considered as a random variable (an uncertain variable with probabilistic distribution), and they were assumed to be independent of each other. This reduces the computational complexity, while circuit timing is estimated conservatively. To capture the correlation between the random variables, various techniques have been proposed; these produce less conservative estimates at the cost of higher computation time [40]. The canonical delay model proposed in [71] directly represents the underlying parameters that affect gate delays and wire delays as random variables, and all timing quantities as a linear affine form of the variables. This technique became popular for its low computational overhead for capturing the correlation between gate and wire delays.

Despite these efforts to reduce the conservatism of the estimates, the number of uncertain parameters, or the conservatism induced by the uncertain parameters, is expected to increase significantly in the near future. Random dopant fluctuations are estimated to affect about 5% (STD/MEAN) of gate delays, about 11% of setup times, and up to 25% of clock-to-output delays in 70nm devices [49]. The amount of variation is expected to be greater in 32nm node or beyond because a smaller number of dopants dominate the threshold voltage. Line-edge roughness also adds a certain amount of variation to each gate delay. To capture these types of variations precisely, timing analysis is required to handle the number of parameters greater than the size of a circuit. In multi-corner static timing analysis, such a large number of random sources is not easily addressed [38, 29]. Block-based SSTA that efficiently captures

the variations usually ignores topological correlation between arrival times, since high computational complexity is required for dealing with the complex correlation between that many random variables, again adding conservatism to the estimation.

There exists some previous research taking the topological correlation into account. In [20], reconvergent fanouts that cause topological correlation are detected through dependency lists. The dependency list of a gate contains all the gates whose arrival times affect the arrival time of the gate. Each input of a gate has a dependency list propagated from the fan-in of the gate, and the common gates in the dependency lists (common supporting gates) are identified. The arrival time of the gate is derived from the common gates. Since the dependency lists can consume a large amount of memory, it takes a user-defined parameter to specify the size of dependency lists or to carry forward the n most recent gates where n is also user-defined. An algorithm proposed in [4] selects a specific type of gate out of common supporting gates of each gate. From the selected gates, topological correlation can be effectively taken into account using an enumeration technique based on conditional probabilities. The worst case computational complexity is exponential, so the authors in [4] selectively perform the enumeration. In [6], an exact algorithm is proposed which interprets a timing graph as a Bayesian network. While the worst case run time is still exponential, it grows with the largest clique size rather than the circuit size. Recently, an approach was proposed in [87, 84] which captures topological correlation by extending the popular canonical delay model. Since

it requires large amounts of memory to capture random sources whose number is greater than the circuit size as the canonical form, the authors also proposed a pruning technique in [88].

The disadvantages of the previous approaches can be summarized as follows. The exact methods in [4, 6] require exponential run time in the worst case. Even approximation methods in [4, 20, 87] require large amounts of memory to maintain dependency lists (the extended canonical form in [87] is similar to the dependency lists). Actually, many techniques to handle reconvergent fanouts in a graph including ones to simply detect them, use dependency lists and have large memory requirements [80, 59].

The major contributions of this chapter can be summarized as follows.

- We show that a timing graph has a hierarchy of specially defined subgraphs, called ellipse graphs. It provides a new interpretation of timing graphs.
- We propose a new directed acyclic graph search algorithm, which allows us to traverse the ellipse graphs. Using the algorithm, we can deal with reconvergent fanouts without dependency lists in linear space complexity.
- We interpret a timing graph as an algebraic factored form and propose an algorithm that modifies factors in the expression without expansion. Thus it can explore numerous equivalent algebraic expressions efficiently. This can be considered as a generalization of the techniques proposed in [4, 6].

- We propose various criteria to determine an algebraic expression that minimizes the topological correlation error.

2.2 Preliminaries

A *literal* is an atomic formula (atom). A *max-plus expression* (MPE) is an algebraic expression made up of addition operations, maximum operations and literals. The number of literals in an MPE f is denoted by $\rho(f)$. Note that maximum and addition have the following algebraic properties.

- *Associative*

  $\max(\max(x, y), z) = \max(x, \max(y, z))$

- *Commutative*

  $\max(x, y) = \max(y, x)$

- *Distributive*

  $x + \max(y, z) = \max(x + y, x + z)$

Note that only addition is distributive over maximum. Let $\cdot$ denote maximum operation for simplicity. For example,

$$x + \max(y, z) = x + y \cdot z = (x + y) \cdot (x + z)^1.$$

A timing graph is a directed acyclic graph (dag) $G = (V, E)$ where each edge e is associated with a delay variable d_e. Each vertex v is associated with a

[1]These notations may be counterintuitive because + is not distributive over $\cdot$ in elementary algebra. It is helpful to consider it as a Boolean expression.

value $\delta(v)$, called a *level*, such that $\delta(w) < \delta(v)$ for all $(w, v) \in E$. A *source* is a vertex with no incoming edge and a *sink* is a vertex with no outgoing edge. A sequence of vertices $(v_1, ..., v_n)$ is a *partial path* if $(v_i, v_{i+1}) \in E$ for all $i = 1, ..., n-1$. A partial path $(v_1, ..., v_n)$ is a *path* if 1) v_1 is a specified vertex or a source and 2) v_n is a specified vertex or a sink. If a vertex v is reachable from a vertex u, then u is a *predecessor* of v and v is a *successor* of u. If there is an edge from u to v, then u is a *direct predecessor* of v, and v is a *direct successor* of u. The sets of direct predecessors and successors of v are denoted by $N^-(v)$ and $N^+(v)$, respectively. An arrival time is a function of delay variables and can be represented by a max-plus expression. The *PERT form* of the arrival time of a vertex v is an MPE corresponding to

$$at(v) = \begin{cases} 0 & \text{if } v \text{ has no incoming edge;} \\ \max\limits_{u \in N^-(v)} \{d_{(u,v)} + at(u)\} & \text{otherwise.} \end{cases}$$

Figure 2.1 shows a simple timing graph and the arrival times represented in the PERT form. The PERT form of $at(v_3)$ has 6 variables and 8 literals.

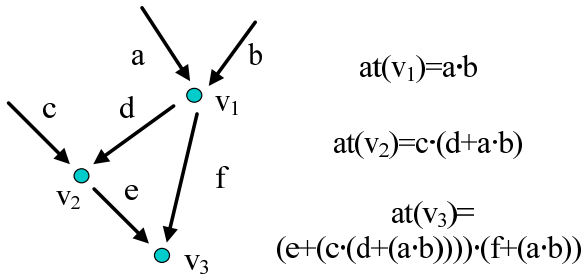


Figure 2.1: In static timing analysis, a dag is topologically sorted, and the arrival time of each vertex is calculated in the topological order.

2.3 Problem Formulation

In conventional static timing analysis, arrival times are obtained by evaluating the PERT forms. Block-based SSTA also uses PERT forms, where variables are random variables. When evaluating a PERT form, block-based SSTA considers all literals to represent different variables, which results in topological correlation error. We will consider an example from Figure 2.1. Suppose that all variables in Figure 2.1 are independent of each other. The 4th and 7th literals in $at(v_3)$ represent the same variable a, but block-based SSTA consider the two literals to represent two independent variables, respectively, and the correlation between the two literals is ignored. If an arrival time is represented as an MPE where the number of literals equals the number of variables, then we can derive the arrival time accurately. This actually happens in tree-structured circuits (i.e., fanout-free circuits). Also, that is the case for $at(v_1)$ and $at(v_2)$, and they do not have topological correlation error. Thus, the ultimate goal of this paper is to find such a MPE for the latest arrival time of a general circuit.

Due to the algebraic properties, there are a number of MPEs equivalent to a PERT form. For example, using the distributivity, we can find MPEs equivalent to the PERT form of the arrival time of v_3 as follows.

$$at(v_3) = (e + (c \cdot (d + (a \cdot b)))) \cdot (f + (a \cdot b)) \tag{2.1a}$$

$$= (e + c) \cdot (e + d + (a \cdot b)) \cdot (f + (a \cdot b)) \tag{2.1b}$$

$$= (e + c) \cdot (((e + d) \cdot f) + (a \cdot b)) \tag{2.1c}$$

Even if there are numerous, equivalent MPEs, the desired MPE does not exist for the latest arrival time in a general circuit. Nevertheless, we observe that some MPEs produce significantly less topological correlation error than others. Thus, if we can explore equivalent MPEs and can select good one intelligently, we can minimize the error. The exploration process is called *refactoring*, and in this chapter, we propose efficient refactoring algorithms that, for the latest arrival time of a given time graph, find an equivalent MPE which produces less topological correlation error.

2.4 Refactoring Algorithms

In this section, we first present a refactoring algorithm that does not require delay information, called *static refactoring*. If there is only one literal representing a variable in an MPE, the literal is *original*. If there are two or more literals for a variable, one is original and the others are *replicated*. For example, Eq.(2.1a) has 2 replicated literals. Assuming each replicated literal causes the same amount of error, the static refactoring uses the number of replicated literals as a measure of error. Thus, Eq.(2.1c), which has 1 replicated literals, is better than Eq.(2.1a) in the static-refactoring sense. Therefore, we try to find an MPE with the minimum number of replicated literals. Since the number of the original literals is invariant in equivalent MPEs, we simply use the total number of literals as the measure.

2.4.1 Division

In multi-level synthesis, there have been a number of studies on factoring algorithms, which minimize the number of literals in an algebraic expression using the distributive property [8]. They are based on an operation, called *division*, which is defined as follows in our notation. Division is an operation which, given MPEs f and p, find MPEs q and r such that $f = (p + q) \cdot r$. We say that the division of *dividend* f by *divisor* p generates *quotient* q and *remainder* r. For example,

$$f = (a \cdot b + c) \cdot (a \cdot b + d) \cdot (e + h)$$

can be divided by $a \cdot b$ and be expressed as follows:

$$f = (a \cdot b + c \cdot d) \cdot (e + h).$$

Note that only 6 literals are required rather than 8 literals. The factoring algorithms have a typical form. Given an expression, they find a good divisor, by which the expression is divided. This process recursively continues on the divisor, the quotient and the remainder. In this sense, the expression is maximally factored. For an MPE of the latest arrival time of a circuit, we may be able to apply one of the algorithms. However, since they take a maximally expanded form as input (they are not refactoring but factoring algorithms), it is required to enumerate all paths in the circuit. Moreover, they can hardly handle the number of literals and terms in the expanded form. Thus, we need to develop a new algorithm for refactoring.

2.4.2 Ellipse Graphs

Definition 1. *If all paths starting from a vertex s contain a vertex t, then t is an antipodal vertex of s.*

Definition 2. *The antipode t of a vertex s is an antipodal vertex of s such that $\delta(t) < \delta(u)$ for all antipodal vertices $u \neq t$ of s.*

Definition 3. *Let $G = (V, E)$ be a dag with a single source and a single sink and $s \in V$ be a vertex such that $|N^+(v)| \geq 2$. Let t be the antipode of s. Then, the ellipse graph of s, s-ellipse graph, is a subgraph $G' = (V', E')$ of G such that 1) V' and E' are the set of vertices and edges, respectively, reachable from s, but not from t in G and 2) V' contains t.*

We can define an ordering relation for $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ as follows. $G_1 \leq G_2$ if and only if $V_1 \subseteq V_2$ and $E_1 \subseteq E_2$. The set of the ellipse graphs in G become a partially ordered set. An ellipse graph G_1 is a *parent (ellipse) graph* of G_2 if $G_2 < G_1$ and there is no smaller ellipse graph than G_1 in the circuit graph.

Theorem 1. *Let $G = (V, E)$ be a dag with a single source and a single sink and $s, v \in V$ be vertices with more than one edge such that v is reachable from s. Also let G_s and G_v be the ellipse graphs of s and v, respectively. Let t and w be the antipodes of s and v, respectively. If v is in G_s, then $\delta(w) \leq \delta(t)$ and $G_v < G_w$.*

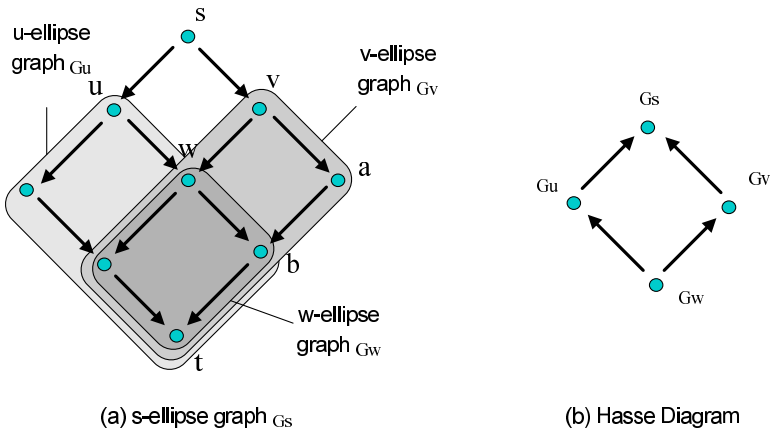


Figure 2.2: Ellipse graphs and the Hasse diagram representing their partial order

Proof. If $w = t$, $\delta(w) = \delta(t)$. We consider the case $w \neq t$. In G, there exists a path $(v, ..., w)$ that does not contain t. Otherwise, w is not the antipode of v. Suppose that $\delta(w) > \delta(t)$. Then, there exists a path $r = (s, ..., v, ..., w, ...)$ in G such that $t \notin r$. This is a contradiction. All vertices and edges in G_v are reachable from s, but they are not reachable from t since $\delta(w) \leq \delta(t)$. Thus $G_v < G_w$. □

Figure 2.2 shows an ellipse graph and its descendant ellipse graphs. The set of all the ellipse graphs is a partially ordered set and can be represented as a Hasse diagram.

2.4.3 Topological-order First Search

In this subsection, we propose an algorithm, topological order first search (TFS), to traverse an ellipse graph. The TFS takes a vertex s as

input in a dag with a single source and a single sink. Prior to start of TFS, the graph is topologically sorted, and the levels are used as keys in the heap Q of TFS.

Algorithm 1 TFS († TFS*)

Input: A vertex s in a dag G with a single source and a single sink
Output: The antipode of s (†the sink of G)

1: $v \leftarrow s$, $Q \leftarrow \emptyset$
2: **while** $v = s$ or $Q \neq \emptyset$ († v is not the sink) **do**
3: **for** each $u \in N^+(v)$ **do**
4: insert u to Q if discovered for the first time
5: **end for**
6: $v \leftarrow EXTRACT - MIN(Q)$
7: **end while**
8: **return** v

Lemma 1. *Let v be the vertex being visited and u be a vertex to be visited after v by TFS*. Then, $\delta(u) \geq \delta(v)$.*

Proof. In order for u to be visited after v, u is in the heap Q or u is a successor of a vertex in the heap Q. If u is in the heap Q, $\delta(u) \geq \delta(v)$ by the heap property. If u is a successor of a vertex w in the heap Q, then $\delta(u) > \delta(w) \geq \delta(v)$ by the topological sorting and the heap property. □

Lemma 2. *Let v be a vertex being visited by TFS* running on a dag $G = (V, E)$ from a vertex $s \in V$. Then, v is an antipodal vertex of s if and only if $v \neq s$ and the heap Q is empty.*

Proof. We first prove the "if" part. Let t be the sink of the given graph G. Note that t is an antipodal vertex of s since the given graph G has one sink.

Suppose that v is not an antipodal vertex of s. Then, since $v \neq t$ and $v \neq s$, $\delta(s) < \delta(v) < \delta(t)$ and there exists a path $p = (u_0 = s, ..., u_n = t)$ such that $v \notin p$. Let i be the largest integer such that $\delta(u_i) < \delta(v)$. Note that u_i is visited before v by Lemma 1 and u_{i+1} is discovered. If it is the first time to discover u_{i+1}, u_{i+1} is inserted to the heap. Otherwise, u_{i+1} sits in the heap. Note that $\delta(u_{i+1}) \geq \delta(v)$. If $\delta(u_{i+1}) = \delta(v)$ and u_{i+1} is extracted first, then u_{i+2} is discovered and sits in the heap when v is being visited. If $\delta(u_{i+1}) = \delta(v)$ and v is extracted first, then u_{i+1} is in the heap. If $\delta(u_{i+1}) > \delta(v)$, v is extracted first and u_{i+1} is in the heap. Thus the heap is not empty when v is being visited. Let us now prove the "only if" part. Suppose the heap is non-empty and let h be an element in the heap. Then $\delta(v) \leq \delta(h)$ by the heap property. If we add $\pi[u] \leftarrow v$ statement before the insertions to Q, it shows a path from s to h, which does not contain v because the π assignment for v cannot be executed before. If a path exists such that $(s, ..., h, ..., v, ...)$, then $\delta(h)¡\delta(v)$ by the topological sorting. This is a contradiction and there is no such path. Therefore, a path exists that does not contain v. □

Theorem 2. *Given a dag $G = (V, E)$ with a single source and a single sink and a vertex $s \in V$ such that $|N^+(s)| \geq 2$, TFS traverses the s-ellipse graph in the topological order.*

Proof. Obvious from Lemma 1 and Lemma 2. □

2.4.4 Static Refactoring

In this subsection, we explain a static refactoring algorithm in detail. The algorithm takes a timing graph as input. In the beginning, we add a *supersource* and an edge with zero delay from the supersource to each source. We also add a *supersink* and an edge with zero delay from the supersink to each sink. Then all paths in the timing graph start from the supersource and end in the supersink. We call the modified timing graph the *root graph*. The root graph is topologically sorted and REFACTOR starts with the supersource. Without loss of generality, we assume the ellipse graph of the supersource is the root graph. If we add an edge with zero delay from the supersource to the supersink, the assumption becomes true, but it may degrade the quality of results. Thus, we can use the following algorithm instead: if the sink of the ellipse graph being traversed by REFACTOR is not the supersink, we continue to call REFACTOR with the sink until it becomes the supersink, and then add all the results up which results in the latest arrival time.

If the variable *divide* is false for all vertices, REFACTOR performs TFS, and hence traverses the root graph in the topological order, producing the arrival times in the PERT form. Let v be a vertex with more than one outgoing edge in an ellipse graph G and u be the antipode of v. Let f_v and f_u be MPEs of the arrival times of v and u, respectively. If we divide f_u by f_v, f_u can be represented as follows.

$$f_u = (f_v + q) \cdot r$$

We can generate q and r by using the following division algorithm. Since f_u is the maximum of the delays of paths that end in u, we can partition the paths into the two path sets P_h and P_r, where P_h contains paths that pass through v and P_r contains the others. Let P_q be a path set that contains all paths between v and u. The MPE of the delay of a path p is denoted by d_p. Then,

$$f_u = \prod_{s \in P_h} d_s \cdot \prod_{p \in P_r} d_p = (f_v + \prod_{s \in P_q} d_s) \cdot \prod_{p \in P_r} d_p$$

Therefore, an MPE of the latest arrival time of the v-ellipse graph can be q, and if v is eliminated from G, an MPE of the arrival time of u can be r. Figure 2.3(a) and (b) illustrates the division f_u by f_v in a graph.

This division algorithm is implemented as follows. Suppose that REFACTOR runs on the ellipse graph G and encounter v. If *divide* is true for v, REFACTOR excludes the delays of the paths that pass through v by setting $at[v] = NIL$. Then, the arrival time of u equals r. In addition, it creates a temporary vertex whose arrival time is $(f_v + q)$ and connects it to u to perform the max operation on r and $(f_v + q)$, making f_u into the divided form. If REFACTOR encounters another vertex with more than one outgoing edge before it reaches u, it can perform a division operation in the same way. Thus, if the temporary vertex is not considered, f_u becomes an optimized (refactored) form of r. To generate q, it is necessary to traverse the v-ellipse graph separately. Note that an ellipse graph is a dag with a single source and a single sink. Besides, since it is already topologically sorted, we can recursively call REFACTOR, which produces an optimized (refactored) form of q. In this

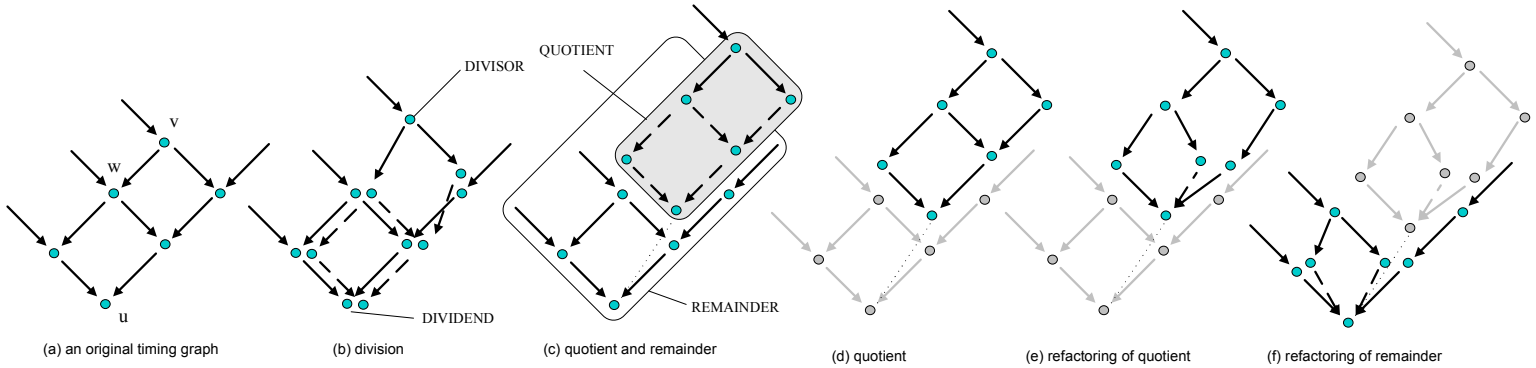


Figure 2.3: The implication of division in a timing graph and recursive refactoring example

sense, f_u is maximally refactored similar to typical factoring algorithms. Once an ellipse graph is separately traversed, the result is stored and used again. If the Hasse diagram is considered that represents the partial order of all ellipse graphs in the root graph, REFACTOR can be deemed to perform the depth first search in the Hasse diagram.

While REFACTOR traverses the v-ellipse graph G_v, if it encounters a vertex w with more than one outgoing edge, it can traverse the w-ellipse graph, G_w. Then G_v is a parent graph of G_w. Consider an edge e in G_w. If in G_v, all the paths that contain e pass through w, e is a *free edge* of G_w. The literal corresponding to the edge is a *free literal*. Thus, given an ellipse graph, its free literals depend on its parent graph. For example, in Figure 2.3(e), the v-ellipse graph is the parent graph of the w-ellipse graph, which has 3 free literals (line arrows), while it has 2 free literals when the parent graph is the root graph as shown in Figure 2.3(f).

If REFACTOR divides f_u by f_v, replicated literals of some original literals in f_u can increase and we call the increase *cost*. Similarly, replicated literals of some original literals can decrease and we call the decrease *revenue*. Depending on cost and revenue, $\rho(f_u)$ can increase or decrease with the division. Thus we need to estimate $profit(v)$, the decrease of $\rho(f_u)$ by the division, and REFACTOR perform the division only if $profit(v) > 0$. We consider $\rho(q)$ as cost, but we can subtract the number of the free literals. The number of free literals of a ellipse graph cannot be stored with the refactoring result of the ellipse graph because it depends on the parent graph. Thus we conservatively

Algorithm 2 REFACTOR

Input: A vertex s in a dag G with a single source and a single sink, $refactored$
Output: The latest arrival time in the s-ellipse graph

$v \leftarrow s$
$Q \leftarrow \emptyset$
$at[s] \leftarrow 0$
while $v = s$ or $Q \neq \emptyset$ **do**
  $divide \leftarrow false$
  if $v \neq s$ and $|N^+(v)| \geq 2$ **then**
    if $refactored[v] = NIL$ **then**
      $REFACTOR(v, refactored)$
    end if
    $(antipode, propat) \leftarrow refactored[v]$
    if $profit(at[v], propat) > 0$ **then**
      $divide \leftarrow true$
    end if
  end if
  if $divide = true$ **then**
    $at[v] \leftarrow NIL$
    $at[v_t] \leftarrow propat$ where v_t is a new vertex and $d_{(v_t, antipode)} = 0$
    $FI_t[antipode] \leftarrow FI_t[antipode] \cup v_t$
    insert $antipode$ to Q if discovered for the first time
  else
    for each $u \in N^+(v)$ **do**
      insert u to Q if discovered for the first time
    end for
  end if
  $v \leftarrow EXTRACT - MIN(Q)$
  $at_t \leftarrow 0$
  for each $u \in N^-(v) \cup FI_t[v]$ **do**
    $e \leftarrow (u, v)$
    if $at[u] \neq NIL$ **then**
      $at_t \leftarrow MAX(at_t, d_e + at[u])$
    end if
  end for
  $at[v] = at_t$
end while
$refactored[s] \leftarrow (v, at[v])$
return $at[v]$

estimate the number of the free literals by assuming the parent graph is the root graph. We estimate revenue as follows. If the division is not performed on v but on all subsequent vertices with more than one outgoing edge, f_v occurs $|N^+(v)|$ times in f_u. If the division is performed on v, f_v occurs only once in f_u. Thus we consider $(|N^+(v)|-1) \times \rho(f_v)$ as revenue. Therefore, we estimate the decrease of $\rho(u)$ by the division f_u over f_v as follows.

$$profit(v) = (|N^+(v)| - 1) \times \rho(f_v) - \rho(q) + k \tag{2.2}$$

where k is the number of free literals of the v-ellipse graph when the parent graph is the root graph.

2.4.5 Dynamic Refactoring

In this subsection, we present a refactoring technique that requires delay information, thereby called *dynamic refactoring*. The lower and upper bounds of d_e are denoted by d_e^{min} and d_e^{max}, respectively. Similarly, the lower and upper bounds of an MPE f are denoted by f^{min} and f^{max}, respectively. The lower and upper bounds of the arrival time of a vertex v are denoted by $at_{min}(v)$ and $at_{max}(v)$, respectively. We can obtain the *observable bound* $b(v)$ of a vertex v as follows.

$$b(v) = \begin{cases} at_{min}(v) & \text{if } v \text{ has no outgoing edge;} \\ \max(at_{min}(v), \min\limits_{u \in N^+(v)} \{b(u) - d_{(v,u)}^{max}\}) & \text{otherwise.} \end{cases}$$

An edge $e = (w, v)$ is *unobservable* if $at_{max}(w) + d_e^{max} < b(v)$. A vertex is unobservable if all outgoing edges of the vertex are unobservable. A graph can be *reduced* by eliminating all unobservable edges and vertices. It is important

to note that the reduction can be applied to all ellipse graphs as well as the root graph. Therefore, we use TFS to calculate the lower and upper bounds of the arrival times in the topological order in an ellipse graph. Also we use another TFS to calculate reachable bounds in the reverse-topological order and determine unobservable vertices and edges. Those two procedures are added at the beginning of REFACTOR, and the ellipse graph is reduced. In the reduced graph, we can use Equation(2.2) (*dynamic-lits*). However, in order to use observable bounds more aggressively, we propose another profit criterion. We define the *spread* $\Lambda(f)$ of an MPE f as follows.

$$\Lambda(f) = f^{max} - f^{min}$$

We call the spread of the latest arrival time the *output spread*. We can obtain the *observable spread* $\Lambda^o(f, v)$ of an MPE f and a node v as follows.

$$\Lambda^o(f, v) = \begin{cases} f^{max} - \max\left(f^{min}, b(v)\right) & \text{if } f^{max} > b(v); \\ 0 & \text{otherwise.} \end{cases}$$

If the arrival time of a vertex v is represented by an MPE f, the observable spread $\Lambda^o(f, v)$ can be considered as the output spread caused by the spread of the arrival time of v. We estimate the decrease of the output spread by the division f_u over f_v as follows.

$$profit(v) = (|N^+(v)| - 1) \times \Lambda^o(f_v, v) - \Lambda^o(f_v^{max} + q, u)$$

where u is the antipode of v and q is the refactored form of the latest arrival time of the v-ellipse graph.

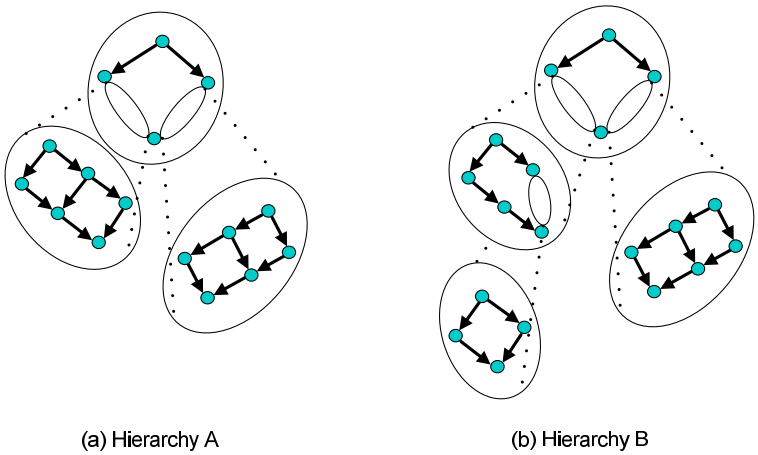


Figure 2.4: Refactoring takes a flat timing graph as input and generates a hierarchy of ellipse graphs.

2.5 The Hierarchical Timing Graph

The result of refactoring can be represented in a hierarchical timing graph where each component is an ellipse graph. Since each MPE corresponds to a hierarchical timing graph, there are numerous, equivalent hierarchical timing graphs. REFACTOR explores a part of them and produce a "good" hierarchy in the sense implied by the given profit function. For example, REFACTOR explores 9 possible hierarchies when the timing graph in Figure 2.2 is given as input, and Figure 2.4 shows two among them. The selection of one among the 9 possible hierarchies is determined by the profit function. We believe that this new point of view for a timing graph can foster several applications in which the following properties may be used:

- Each component in the hierarchy becomes tree-like as a result of the effort for dealing with the problem caused by re-convergent paths. Many

algorithms working on a graph can produce the optimal solution if reconvergent paths do not exist (i.e., the input is a tree). Thus, refactoring may be able to enhance the quality of results in the algorithms.

- Hierarchical analysis is a general techniques for dealing with the complexity. Each component in the hierarchical timing graph is simple, and we may be able to perform a more expensive but accurate analysis for them.

- The more stages a signal goes though, the more uncertain the absolute timing value of the signal is. Thus, it is very difficult to predict the interactions of signals (e.g., MIS and crosstalk) in the middle of circuits. In each component of the hierarchy, we can perform more accurate analysis using the relative timing values from the common starting point. Also, in SSTA, the arrival times become contaminated by the linear approximation error of the max operator as they propagate. In the hierarchical timing graph, the analysis is started over in each component, and the accuracy can be improved.

2.6 Notes on Implementation

Static timing analysis needs to consider both rising and falling transitions. To deal with this, we run a conventional static timing analysis algorithm on a given timing graph but instead of computing rising and falling arrival times with direct predecessors and incoming edges, we create new vertices and

edges corresponding to them which compose a new timing graph with twice as many vertices as the original one. We can simply run REFACTOR on the new timing graph, and then rising and falling transitions can be considered easily. We have proposed REFACTOR as a recursive function to explain the concept. However, it is not necessary to implement it as a recursive function. We can enumerate vertices in reverse topological order. During the enumeration, if a vertex v with more than one outgoing edge is encountered, the v-ellipse graph is traversed by calling REFACTOR. If another vertex w with more than one outgoing edge is encountered during the traversal, the w-ellipse graph is already refactored since w comes first before v in the reverse-topological order. Thus REFACTOR is not called recursively. This implementation is usually more efficient in both CPU time and memory than the recursive version.

2.7 Exploring More Candidate Solutions

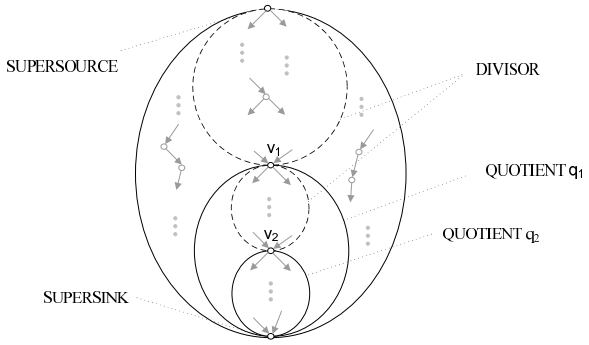


Figure 2.5: Divisors may not be re-factored if many antipodes are the supersink.

In every division operation of Algorithm 2, the dividend should be the

antipode of the divisor, which allows us to have developed the simple refactoring algorithm. However, this constraint reduces the solution space that are explored by the refactoring algorithm. In theory, it is possible to perform the division between any two vertices, which may be able to enhance the algorithm. However, this may complicate the algorithm, increasing the runtime. Thus, in this section, we propose a method that can enhance REFACTOR with a slight modification.

According to our investigation to several benchmark circuits, the antipodes of many vertices are actually the supersink in the timing graph, and Figure 2.5 illustrates how refactoring is performed in this case. In Figure 2.5, the antipodes of both v_1 and v_2 are the supersink. Suppose the following scenario: we divide the supersink by v_1, which results in the quotient q_1. Then, in order to optimize the quotient q_1, we divide the supersink by v_2. Then we continue to examine the quotient q_2 for possible division.

From this example, we can notice that REFACTOR can optimize quotients well in a recursive manner, but divisors, represented by dotted ellipses in Figure 2.5, may remain unoptimized. In particular, this is more problematic in circuits with a high logic depth because the depth of the recursive refactoring is also deep and a large part remains as it is. Thus, whenever REFACTOR performs division, we optimize the divisor using a dedicated algorithm. The basic concept of the algorithm is as follows. Conceptually, we turn the subgraph corresponding to the divisor upside down and perform refactoring in the same way as REFACTOR except the fact that dividends should be the sink

of the sub-graph.

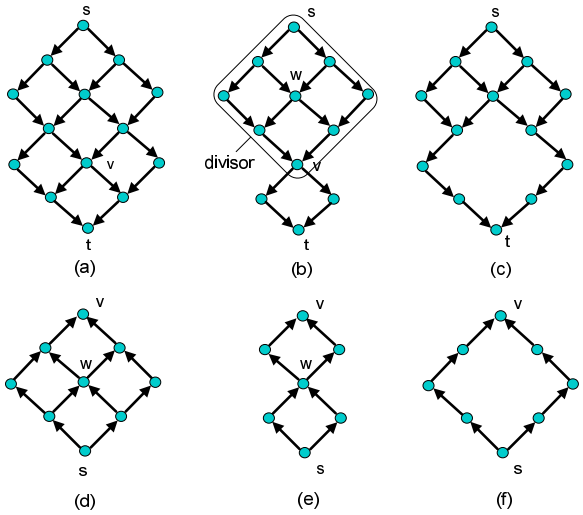


Figure 2.6: An example of divisor refactoring

Figure 2.6 illustrates the proposed approach. Suppose that REFACTOR takes the timing graph in Figure 2.6(a) as input and decides to divide the vertex t by the vertex v. Then, the graph corresponding to the divisor and the quotient is shown in Figure 2.6(b), and the one corresponding to the reminder is shown in Figure 2.6(c). In Figure 2.3, we have focused the quotient and the reminder, but our focus at this time is the divisor, the arrival time of v. The graph representation of the divisor is surrounded by a box in Figure 2.6(b). The representation of the divisor is overturned which is shown in Figure 2.6(d). REFACTOR with the restriction, which will be denoted by DIVREFACTOR later, is started from v, and in all divisions, the dividend should be s. Suppose that s is divided by w. Then, the quotient and the divisor are represented as in Figure 2.6(e), and the remainder is represented

Algorithm 3 DIVREFACTOR

Input: a dag G with a single source s and a single sink t, at
Output: The latest arrival time in the s-ellipse graph

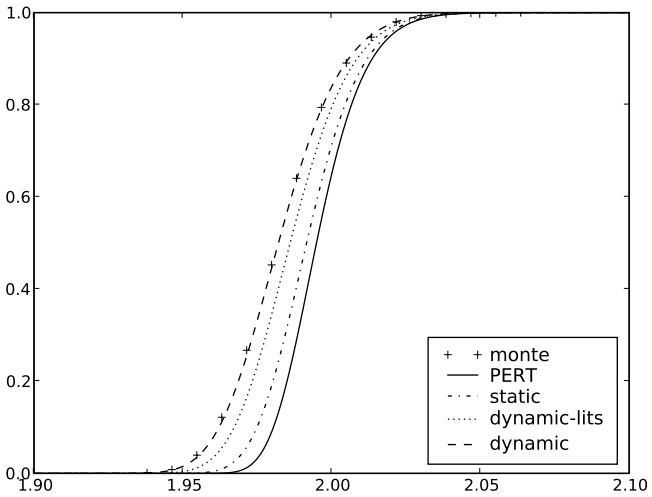
1: $v \leftarrow t$
2: $Q \leftarrow \emptyset$
3: $rev\_at[t] \leftarrow 0$
4: **while** $v = t$ or $Q \neq \emptyset$ **do**
5: $\quad divide \leftarrow false$
6: $\quad$ **if** $v \neq t$ and $|N^-(v)| \geq 2$ **then**
7: $\quad\quad propat \leftarrow rev\_at[v] + at[v]$
8: $\quad\quad$ **if** $profit(rev\_at[v], propat) > 0$ **then**
9: $\quad\quad\quad divide \leftarrow true$
10: $\quad\quad$ **end if**
11: $\quad$ **end if**
12: $\quad$ **if** $divide = true$ **then**
13: $\quad\quad rev\_at[v] \leftarrow NIL$
14: $\quad\quad rev\_at[v_t] \leftarrow propat$ where v_t is a new vertex and $d_{(v_t,s)} = 0$
15: $\quad\quad FO_t[s] \leftarrow FO_t[s] \cup v_t$
16: $\quad\quad$ insert s to Q if discovered for the first time
17: $\quad$ **else**
18: $\quad\quad$ **for** each $u \in N^+(v)$ **do**
19: $\quad\quad\quad$ insert u to Q if discovered for the first time
20: $\quad\quad$ **end for**
21: $\quad$ **end if**
22: $\quad v \leftarrow EXTRACT - MIN(Q)$
23: $\quad at_t \leftarrow 0$
24: $\quad$ **for** each $u \in N^+(v) \cup FO_t[v]$ **do**
25: $\quad\quad e \leftarrow (u, v)$
26: $\quad\quad$ **if** $rev\_at[u] \neq NIL$ **then**
27: $\quad\quad\quad at_t \leftarrow MAX(at_t, d_e + rev\_at[u])$
28: $\quad\quad$ **end if**
29: $\quad$ **end for**
30: $\quad rev\_at[v] = at_t$
31: **end while**
32: **return** $rev\_at[v]$

as in Figure 2.6(f).

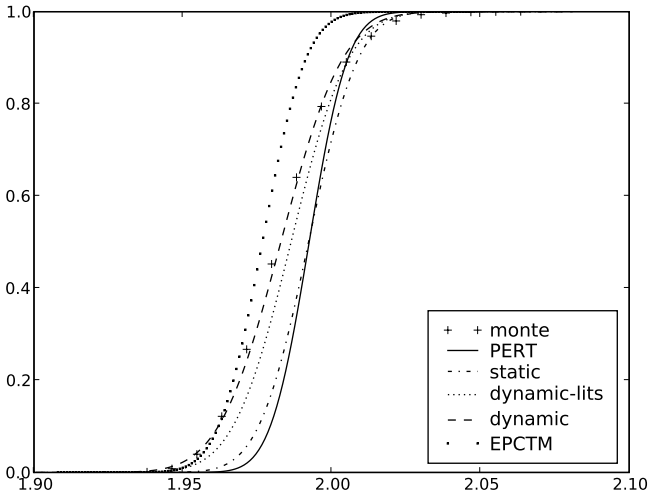
The overall procedure is summarized in Algorithm 3. In a run of REFACTOR that is started from a vertex s, if we decide to perform division at a vertex v, we can invoke DIVREFACTOR to optimize the divisor further, and the subgraph between the vertex s and v (i.e., the graph corresponding to the divisor) becomes the input for DIVREFACTOR. In DIVREFACTOR, we traverse the input graph backward (i.e., reverse topological order). Thus, for the heap Q, the reverse topological order is used as the key (line 22). In REFACTOR, before actual division is performed for each vertex, we pre-compute the quotient and store the sum of the quotient and the corresponding divisor into $refactored$. The sum is denoted by *propat* in REFACTOR. In DIVREFACTOR, if division is performed at a vertex v, the quotient becomes $at[v]$, which is already computed in REFACTOR before DIVISOR_REFACTOR is invoked. Thus *propat* is calculated as line 7. Unlike REFACTOR, this procedure is not done recursively, and the dividend is always the source s of the input graph.

2.8 Experimental Results

We have implemented the proposed algorithm in C++, and have used ISCAS85 circuits as benchmarks. All experiments were performed on a 3.0GHz 2 Xeon X5450 Linux machine. All benchmarks are technology-mapped to the TSMC 180nm standard cell library, and the nominal delay annotated to each edge in the timing graph is the sum of the gate and wire delays obtained from the Standard Delay Format (SDF) file.



(a) Discrete PDF SSTA



(b) Canonical SSTA

Figure 2.7: The c.d.f of c499 for $\gamma = 20\%$ shows the relative performance of various criteria of refactoring.

Table 2.1: Numbers of literals ($\gamma = 8\%$)

	static		**dynamic-lits**	
ckt	PERT	refactor	PERT	refactor
c17	40	40	20	20
c432	280595	4440	112630	178
c499	808784	19208	119640	3005
c880	29508	8438	898	76
c1355	808688	16360	134312	3281
c1908	4865369	42059	1485010	4092
c2670	129076	13785	9263	308
c3540	13688338	72487	904875	4425
c5315	1582121	53288	238192	3401
c6288	$7.4985 \text{x} 10^{16}$	100488501	$9.1431 \text{x} 10^{15}$	5373024
c7552	1479776	79065	9494	227

In this experiment, global and spatially correlated variation are not considered (i.e., the number of global sources of variation is zero in canonical first-order forms). The delay of each edge has the independent variation following a Gaussian distribution, and the 3σ point is γ of the nominal delay (i.e., the sensitivity coefficient to the independent variation is $\gamma\mu/3$ in canonical first-order forms).

Table 2.1 shows the numbers of literals of the PERT (original) and refactored forms for the latest arrival time of each benchmark circuit. The static refactoring technique is used for the results in the static section. In the dynamic section, the PERT forms are obtained from the reduced root graph, and the refactored forms are optimized for the number of literals (dynamic-lits).

Figure 2.7(a) compares various criteria of refactoring in a discrete PDF

Table 2.2: Comparison with existing methods for ISCAS85 benchmarks (γ=20%)

ckt	monte		Canonical Model [71]				EPCTM [87]				refactoring + [71]				vs. EPCTM	
	μ (ps)	σ (ps)	$\Delta\mu$ (ps)	$\Delta\sigma$ (ps)	CPU (s)	mem (MB)	$\Delta\mu$ (ps)	$\Delta\sigma$ (ps)	CPU (s)	mem (MB)	$\Delta\mu$ (ps)	$\Delta\sigma$ (ps)	CPU (s)	mem (MB)	CPU (x)	mem (x)
c17	151.3	4.8	0.4	-0.2	0	1.2	0	-0.3	0	1.3	0.4	-0.2	0	1.2	1.0	1.0
c432	2329.4	31.1	26.7	-10.1	0.002	1.6	-0.6	-2.3	0.028	9.9	0.3	0	0.005	1.5	5.6	6.5
c499	1983.1	17.6	9.9	-7.4	0.004	2.3	-5.9	-6.1	0.178	52.9	0.3	-1.2	0.147	3.0	1.2	17.6
c880	2157.9	23.8	0.6	-1	0.002	2.1	-1.1	-3	0.085	39.8	0.2	-0.1	0.006	1.9	14.2	20.8
c1355	2151.2	21.1	14.2	-9.3	0.004	2.3	-5.5	-7.3	0.178	52.4	0.3	-1.4	0.138	2.9	1.3	18.1
c1908	3179.1	33.6	25.4	-12.9	0.004	2.2	-3.7	-0.2	0.152	47.5	2.3	-3.3	0.057	2.4	2.7	20.1
c2670	2293	27.4	23.5	-9.4	0.007	3.1	-2.3	-2.9	0.468	150.1	4.3	-2	0.029	3.0	16.1	50.2
c3540	3283.8	28.9	13.8	-9.5	0.008	3.4	-2.5	-3.4	0.649	197.7	5	-3.5	0.147	4.9	4.4	40.6
c5315	3031	30.6	21.2	-10.6	0.014	5.0	-0.6	-2.2	1.38	579.1	5.4	-2.8	0.073	4.8	18.9	121.5
c6288	9618.6	61.8	92.3	-30	0.016	6.5	-3.7	-1.3	3.42	1187.4	42.7	-13.6	2.07	48.3	1.7	24.6
c7552	5196.3	60.4	37.6	-25.6	0.012	5.5	-6	2.6	1.6	739.2	4.5	-3	0.044	4.8	36.4	153.0
average			24.1	-11.4	0.006	3.2	-	-	0.739	277.9	5.9	-2.8	0.246	7.1	9.4	43.1

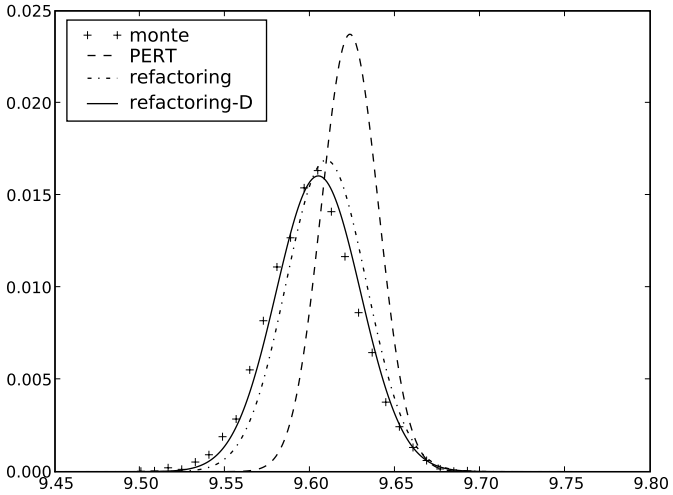
Table 2.3: Comparison with existing methods for ISCAS85 benchmarks (γ=8%)

ckt	monte		Canonical Model [71]				EPCTM [87]				refactoring + [71]				vs. EPCTM	
	μ (ps)	σ (ps)	$\Delta\mu$ (ps)	$\Delta\sigma$ (ps)	CPU (s)	mem (MB)	$\Delta\mu$ (ps)	$\Delta\sigma$ (ps)	CPU (s)	mem (MB)	$\Delta\mu$ (ps)	$\Delta\sigma$ (ps)	CPU (s)	mem (MB)	CPU (x)	mem (x)
c17	150	2.3	0	0	0	1.2	0	0	0	1.3	0	0	0	1.2	1.0	1.0
c432	2311.8	14.3	9.3	-4.4	0.001	1.6	0.2	-0.5	0.027	9.9	0.2	-0.2	0.002	1.5	13.5	6.6
c499	1949.4	8.7	6.3	-3.4	0.004	2.3	-0.6	-2	0.178	52.9	0.6	-0.7	0.058	2.5	3.1	21.5
c880	2145.5	10.3	0.2	-0.2	0.003	2.1	0.2	-1	0.104	39.8	0.3	-0.2	0.003	1.9	34.7	21.2
c1355	2113.6	10.2	7.9	-3.9	0.004	2.3	0	-2.9	0.151	52.4	0.3	-0.6	0.042	2.4	3.6	21.6
c1908	3165.4	15	7.6	-3.9	0.004	2.2	0.2	0.1	0.152	47.5	0.4	-0.1	0.057	2.3	2.7	20.9
c2670	2270.7	12.7	8.4	-4	0.007	3.1	0.2	-0.8	0.469	150.1	1.9	-0.6	0.011	2.7	42.6	54.9
c3540	3263.6	12.5	4.4	-2.4	0.008	3.4	0.1	-0.8	0.649	197.7	1.5	-0.8	0.05	3.4	13.0	59.0
c5315	3018.1	14.5	7.9	-4.3	0.014	5.0	0	-0.5	1.839	579.1	2.7	-2.2	0.037	4.5	49.7	129.8
c6288	9602.1	25.7	21.3	-8.9	0.021	6.5	-0.6	-0.4	4.259	1187.4	7.4	-2.1	2.361	38.1	1.8	31.1
c7552	5186.9	26.6	5.2	-4.8	0.015	5.5	0.1	0	1.96	739.2	0.5	-0.6	0.025	4.6	78.4	160.6
average			7.1	-3.7	0.007	3.2	-	-	0.890	277.9	1.4	-0.7	0.241	5.9	22.2	48.0

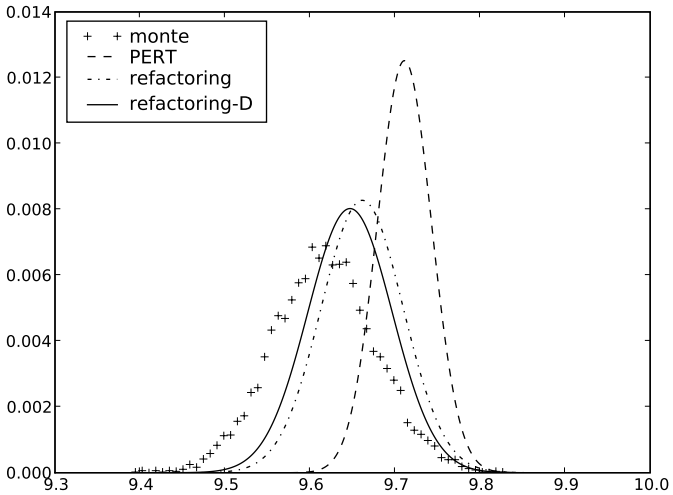
SSTA. For some benchmarks, static refactoring is as good as dynamic refactoring but Figure 2.7(a) shows their relative performances in general. Figure 2.7(b) compares them in the SSTA with canonical delay model [71] and shows some distortion caused by the Gaussian approximation to results of maximum operations. The dynamic refactoring technique significantly improves the accuracy of any arbitrary block-based SSTA.

The refactoring technique is compared against the SSTA with the extended pseudo-canonical timing model (EPCTM) proposed in [87], and Table 2.2 and 2.3 show the cases where γ=20% and 8%, respectively. We have implemented two versions of [87], but since the one using a spare matrix library is worse than the other simple implementation in both CPU time and memory consumption, we present only one version. Ignoring topological correlation increases the means of latest arrival times and decreases the standard deviations as denoted by the minus signs. The EPCTM is optimized for the accuracy and can use less CPU time and less memory if the accuracy is compromised. The refactoring technique preserves the rule of conservative estimation in static timing analysis by construction, while EPCTM can overestimate the topological correlation of arrival times which often results in optimistic latest arrival times, as shown in all benchmark circuits in Table 2.2 and c499, c1908 and c6288 in Table 2.3. Note that the refactoring technique consumes significantly less amount of memory than EPCTM since it has linear space complexity.

Table 2.4 shows the enhancement by DIVREFACTOR. We perform static refactoring for 4 benchmark circuits, and the number of literals and the



(a) γ=8%



(b) γ=20%

Figure 2.8: The PDFs of the latest arrival time of c6288

Table 2.4: Improvement by Algorithm 3

ckt	refactoring		refactoring-D		B/A
	#literals, A	CPU(s)	#literals, B	CPU(s)	
c3540	72487	0.250	65795	1.300	9.23%
c5315	53288	0.258	48612	0.678	8.77%
c6288	100488501	1.684	58044667	13.822	42.24%
c7552	79065	0.447	72819	1.860	7.9%

runtime are shown in the second and the third columns. Then we perform static refactoring where DIVREFACTOR is added (denoted by refactoring-D), and the results are shown in the fourth and fifth columns. The reduction of the number of literals by DIVREFACTOR is shown in the last column. By additionally invoking DIVREFACTOR, we traverse divisors again and explore more candidate solutions. As a result, we optimize the number of literals further at a small cost of runtime. However, compared to the drastic reduction in Table 2.1, the improvement by DIVREFACTOR is marginal for the following reasons:

- Refactoring has eliminated numerous replicated literals already, and the diminishing return is natural. Given that refactoring captures topological correlation efficiently but it cannot be complete, we may be very close to the limit we can reach.

- In the timing graph, the number of literals of arrival times at a low level is usually smaller than that at a high level. Thus, the numbers of replicated literals of divisors are usually small so there is not much room for improvement. However, they are not that small in circuits with

many logic levels. Thus, the reduction in c6288 is decent compared to the other circuits.

The second reason conversely implies that in most circuits, the original REFACTOR produces good results even in the scenario that concerns us in Section 2.7.

Figure 2.8 shows the probability density functions (PDFs) of the latest arrival time of c6288. As demonstrated in Table 2.2 and 2.3, c6288 is the most complex topological correlation, which is very difficult to capture. For this experiment, dynamic refactoring is employed. In a small amount of variation, refactoring combined with DIVREFACTOR results in almost the same PDF as Monte Carlo simulation even for c6288 as shown in Figure 2.8(a). In a large amount of variation, the improvement by DIVREFACTOR becomes noticeable but obtaining the exact PDF in this case is remained as a challenging problem, as shown in Figure 2.8(b).

2.9 Conclusions

We have proposed a novel refactoring algorithm, and our experiments have shown its improvements over an existing algorithm in terms of accuracy, speed and memory consumption. In addition to the improvements in SSTA, the refactoring algorithms provide a new interpretation of timing graphs. We believe that the refactoring algorithms can be applied to more timing-related EDA applications where reconvergent paths degrade the quality of results such as critical path selection on SSTA [76] and timing optimization (e.g. buffer

insertion). If each gate and wire delay varies as design alternatives, then the process that estimates the change of the latest arrival time by a set of design changes is similar to SSTA. Other applications of refactoring may be part of our future work.

Chapter 3

Path Criticality Computation in Parameterized Statistical Timing Analysis

3.1 Introduction

The timing critical path in a chip is of great interest for almost every step involved in producing VLSI chips, from design to testing. Since the critical path changes from die to die in the presence of process variations, potentially critical paths are of importance in a design. During the design phase, a set of potentially critical paths enables path-based SSTA to be performed, which allows us to estimate parametric timing yield more accurately [57, 58]. If the design does not meet a desired yield level, the paths in the set can be optimized. In adaptive body bias or adaptive supply voltage techniques [45], the potentially critical paths are often replicated in order to estimate the minimum operating voltage without modifying the design. At the end points of these paths, we can insert double-sampling flops for dynamic voltage scaling [24] and/or observable points for debugging as in [81]. For post-silicon validation and manufacturing test, these paths can be targeted by an ATPG tool, which can generate the test vectors sensitizing the paths, and the test vectors can be used to determine the operating frequency (*speed-binning*) of the chip or to check if manufactured chips can be operated at the desired frequency.

Better prediction of the critical path provides major benefits in all these applications, but due to increasing process variability it is becoming more difficult to predict a small set of potentially critical paths in the pre-silicon phases. However, the advent of statistical timing analysis provides a tool to deal with the variability [7]. Recent statistical static timing analysis algorithms [60, 40, 10, 71] are able to capture complex correlations of path delays, which offers a chance of efficiently narrowing down potentially critical paths even with the increased variability.

In order to construct a set of potentially critical paths precisely, it is necessary to accurately compute the path criticality, which is the probability that a path becomes the critical path. Several approaches are proposed to calculate the path criticality. In [71], the authors consider the events for each edge in the timing graph that determines the arrival time of the head (the terminal vertex). If these events occur for all the edges on a path, the path becomes critical. Since these events are assumed to be independent, the path criticality is calculated by multiplying the probabilities of the events. However, since the global sources of variation as well as topological correlation make them correlated events, the independent assumption causes quite a large error. To handle this issue, the authors in [83] consider the joint probability for the correlated events, and the probability is evaluated using the *max* operator provided by the SSTA framework. In [77], the path criticalities are obtained by calculating the probability that each path delay is greater than the latest arrival time. If the max operator and the arrival times in SSTA are accurate,

both approaches can provide accurate criticality values. However, current block-based SSTA algorithms compromise the accuracy for reducing run times; they usually ignore topological correlation and use a linear approximation to the nonlinear max operation. Due to these issues, both approaches are still far from computing precise values. Some issues in computing criticality using the max operator are demonstrated in detail in [54].

The major contributions of this chapter can be summarized as follows.

- We propose a novel way to divide all paths in the circuit into several disjoint sets. The partitioning method allows us to easily calculate the complement path delay, the maximum of all the other path delays excluding the delay of a given path. The cutset method, which is widely used for this type of partitioning, has difficulty in calculating the complement path delay [77].

- Unlike [83], our delay model takes random independent variation into account. Under this model, the path criticality becomes more difficult to calculate because topological correlation of the random independent component causes large error. In order to deal with this issue, we perform algebraic elimination in our path criticality formulation, reducing such a type of error.

- Instead of relying on the inaccurate max operator as in [77, 83] or Monte Carlo sampling as in [54], we propose novel, analytic conditioning operation in parameterized SSTA. For this, we also develop a general result

on conditional statistics by extending Clark's work [17]. Unlike [71], our computation method does not use the independent assumption.

The rest of the chapter is organized as follows. Section 1.3 introduces current SSTA techniques. Section 3.2 defines the path criticality and presents issues with the existing work. Section 3.3 presents our proposed method in detail. Section 3.4 discusses the normal approximation error of the proposed method. Section 3.5 shows experimental results, and Section 3.6 concludes the chapter.

3.2 Path Criticality

Let Ω denote the set of all the paths in the circuit. The delay between two vertices (including path delays) is denoted by d.

Definition 4. *A path p is critical if $d(p) \geq d(s)$ for all $s \in \Omega$.*

Definition 5. *The criticality of a path p is the probability that the path p is critical.*

The criticality is denoted by λ. Thus we can write

$$\lambda(p) = P(\bigcap_{s \in \Omega} d(p) \geq d(s)). \tag{3.1}$$

One of the computation methods proposed in [77] obtains the path criticality by

$$\lambda(p) = P(d(p) \geq \max_{s \in \Omega}\{d(s)\}). \tag{3.2}$$

We can obtain $\max_{s\in\Omega}\{d(s)\}$ from SSTA. Then, Eq.(3.2) is evaluated efficiently using Eq. (1.2), and this single comparison directly determines the path criticality. In this method, while the operands of the maximum contain the path delay being compared, this correlation is ignored in the computation. To demonstrate this, we will consider an example from Figure 3.1. For this

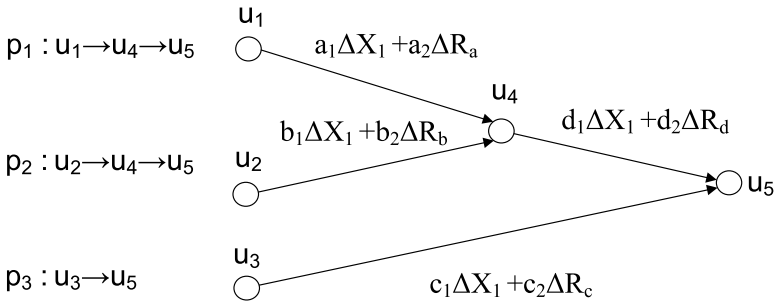


Figure 3.1: A simple timing graph with edge delays of canonical forms. The mean values are assumed zero for illustration purpose.

example, we assume that $a_1 = 0, b_1 = 0, c_1 = 0, d_1 = 0$ and will compute the criticality of p_1. The method of [77] is intended to evaluate

$$\begin{aligned}\lambda(p_1) = P(&\Delta R_a + \Delta R_d \geq \\ &\max\{\Delta R_a+\Delta R_d, \Delta R_b + \Delta R_d, \Delta R_c\}).\end{aligned} \tag{3.3}$$

In parameterized SSTA, the variables on the left hand side in the inequality are lumped into one variable to represent it in the canonical form. Similarly, the maximum on the right hand side is lumped into one variable. So if we let $\Delta R_x = \Delta R_a + \Delta R_d$ and $\Delta R_y = \max\{\Delta R_a + \Delta R_d, \Delta R_b + \Delta R_d, \Delta R_c\}$, then we can write

$$\lambda(p_1) = P(\Delta R_x \geq \Delta R_y). \tag{3.4}$$

While ΔR_x are ΔR_y are correlated (due to the topology), SSTA assumes that they are independent. To reduce this error, the authors in [77] also propose the re-construction method (another method in [77]) which excludes the path delay from the maximum in a constant time. If the re-construction method is used, we can compute the path criticality of p_1 by

$$\lambda(p_1) = P(\Delta R_a + \Delta R_d \geq \max\{\Delta R_b + \Delta R_d, \Delta R_c\}). \tag{3.5}$$

In this case, the left hand side and the right hand side in the inequality becomes less correlated, so the accuracy can be improved. However, it still ignores the correlation induced by ΔR_d. This happens even in the case there is no re-convergence path as in this example.

In addition to ignoring the topological correlation, both methods of [77] heavily rely on the max operator. The max operator in SSTA causes some error to fit the non-linear result of maximum operation into a single linear form. Both the methods represent the maximum of numerous path delays as a single canonical form, and the path criticality depends on the single canonical form excessively, incurring much error. We will consider another example from Figure 3.1 again. For this example, we will assume that $a_2 = 0, b_2 = 0, c_2 = 0, d_2 = 0$. Then, the re-construction method computes the path criticality by

$$\lambda(p_1) = P((a_1 + d_1)\Delta X_1 \geq \max\{(b_1 + d_1)\Delta X_1, c_1 \Delta X_1\}). \tag{3.6}$$

Figure 3.2 illustrates the linear approximation error to the maximum on the right hand side in the inequality. The process that fits the non-linear curve

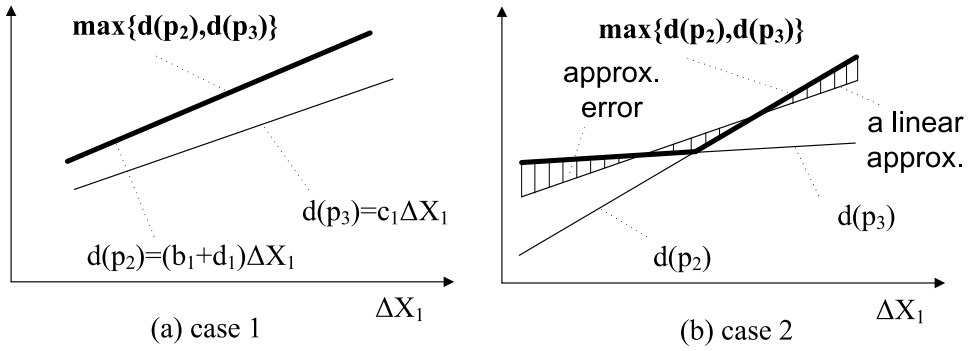


Figure 3.2: Depending on the mean values of $d(p_1)$ and $d(p_2)$ (not shown in the equations) and the values b_1, d_1, c_1, we consider two cases. As in the case 1, if one path delay dominates the other path delay, the approximation is accurate. However, it incurs some error in the case 2.

into a line loses some information. Simply, we can retain all the information using two canonical forms to represent the two path delays. To keep all the information for computing the criticality in this way, we may need as many canonical forms as the number of paths in the circuit. However, since there are usually numerous paths in a circuit, practically it is impossible to use that many canonical forms. Thus, we cannot avoid the use of the max operator completely, but we introduce a new operation, which allows us to compute the path criticality from a desired number of canonical forms.

3.3 Proposed Algorithm

We partition Ω into m groups and compare the delay of the given path to the maximum of all the path delays in each group. Then, the path criticality

can be written as

$$\lambda(p) = P(\bigcap_{i=1}^{m} d(p) \geq U_i) \tag{3.7}$$

where U_i is the maximum of all the path delays in the i-th group. The way that we partition Ω allows us to compute U_i from several arrival times very efficiently. Thus we do not have to enumerate paths to calculate U_i. Instead of computing (3.7) directly, we re-write it by performing algebraic elimination for the m inequalities. This process improves the accuracy by handling topological correlation.

3.3.1 A Simple Timing Graph

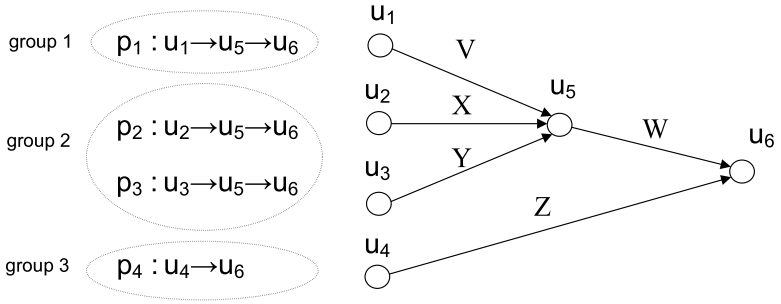


Figure 3.3: A simple timing graph

In this subsection, we demonstrate the first two steps of the proposed algorithm with a simple example. Let us consider a simple timing graph shown in Figure 3.3. In this timing graph, Ω is $\{p_1, p_2, p_3, p_4\}$ which is partitioned into 3 groups. We are to compute the criticality of the path p_1. The comparison with group 1 is trivially true so we can write

$$\lambda(p_1) = P(V + W \geq \max(X + W, Y + W) \cap V + W \geq Z). \tag{3.8}$$

We can re-write it using the distributivity of addition over maximum and performing algebraic elimination as follows.

$$\begin{aligned} & P(V + W \geq \max(X + W, Y + W) \cap V + W \geq Z) \\ & = P(V + W \geq \max(X, Y) + W \cap V + W \geq Z) \\ & = P(V \geq \max(X, Y) \cap V + W \geq Z) \end{aligned} \tag{3.9}$$

As a result, W in the both sides of the first inequality is eliminated. We let $A_1 = V, B_1 = \max(X, Y), A_2 = V + W$ and $B_2 = Z$. Then, we can write the path criticality of p_1 by

$$\lambda(p_1) = P(A_1 \geq B_1 \cap A_2 \geq B_2) \tag{3.10}$$

Note that there is no topological correlation between A_1 and B_1, and between A_2 and B_2. Also note that this algebraic elimination is not done in the runtime but is performed implicitly in the way we formulate the path criticality. This will be clarified in the following subsection.

3.3.2 A General Timing Graph

In this subsection, we show how Ω is partitioned and the algebraic elimination is performed in a general timing graph. Our computation algorithm takes a timing graph as input. In the beginning, we add a *source*, and the source is connected to each primary input via an edge whose delay is the arrival time at the primary input. We also add a *sink*, and the sink is connected to each primary output and each timing test vertex via an edge. The delay of such an edge is the negative of the required arrival time of the corresponding

vertex. Then all paths in the timing graph start from the source and end at the sink. The source and the sink are denoted by v_0 and w, respectively. This timing graph is called an *augmented timing graph* and the construction process from a circuit is well illustrated in [77]. We perform parameterized statistical static timing analysis on the modified timing graph, which produces the arrival time of each vertex. The arrival time is denoted by at. We are given a path $p = (v_0, ..., v_t = w)$ in the timing graph and to compute the path criticality $\lambda(p)$. Then, according to the path p, we will partition Ω (the set of all the paths in the timing graph) into $t + 1$ groups (i.e., $m = t + 1$). Note that the partitioning depends on the given path p, and similar to the algebraic elimination, this partitioning is not done in the runtime and is implicit in our path criticality formulation. The $t + 1$ groups are denoted by $G_0, ..., G_t$. The group G_0 contains the path p. The group G_1 contains all the paths that pass through $(v_1, ..., v_t)$ excluding the paths in G_0. Similarly, the group G_2 contains all the paths that pass through $(v_2, ..., v_t)$ excluding the paths in G_0 and G_1. Generally, for $k = 1, ..., t$, the group G_k contains all the paths that pass through $(v_k, ..., v_t)$ excluding the paths in $G_0, ..., G_{k-1}$ groups. These groups are mutually exclusive and the union of these groups becomes Ω (i.e., the division by these groups is a partition of Ω). This partitioning is illustrated in Figure 3.4. In this example, G_1 does not contain any path since v_1 has only one incoming edge. If we remove all the vertices with only one incoming edge by merging them with their direct predecessor, this case does not happen. Thus, without loss of generality, we assume that every vertex

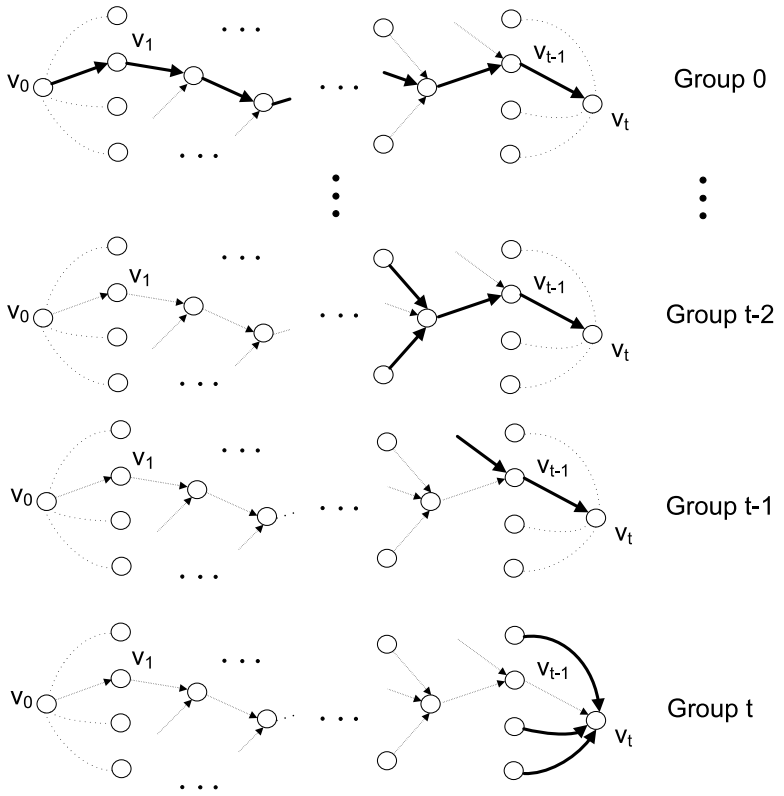


Figure 3.4: Given a path p, we can partition Ω as above. The top figure (group 0) shows the given path p.

has two or more incoming edges (i.e, every group is not empty), and we will write the following derivation without handling such a case specially. Let U_k be the maximum of all the path delays in the group G_k. Then, U_0=$d(p)$ and for $k = 1, ..., t$, we can easily compute U_k by

$$U_k = \max_{u \in N^-(v_k)} \{at(u) + d_{(u,v_k)} + d_{(v_k,...,v_t)} | u \neq v_{k-1}\}. \tag{3.11}$$

where N^- denotes the set of direct predecessors. Using the algebraic elimination as in (3.9), we obtain

$$\begin{aligned} \lambda(p) &= P(\bigcap_{k=0}^{t} d(p) \geq U_k) \\ &= P(\bigcap_{k=1}^{t} A_k \geq B_k) \end{aligned} \tag{3.12}$$

where

$$A_k = d_{(v_0,\ldots,v_k)} \tag{3.13}$$

and

$$B_k = \max_{u \in N^-(v_k)} \{at(u) + d_{(u,v_k)} | u \neq v_{k-1}\}. \tag{3.14}$$

Since we put the path p into a separate group (G_0) and the comparison $d(p) \geq U_0$ is always true, we can exclude the case $k = 0$ in this step.

In parameterized SSTA, since all timing quantities are represented in the canonical form, A_i and B_i in (3.12) are of the canonical form, and the key problem becomes the accurate calculation of the joint probability. This calculation can be done by numerical integration, but the computational complexity of performing multi-dimensional numerical integration is prohibitive. Alternatively, we may compute it by

$$\begin{aligned} P(\bigcap_{i=1}^{t} A_i \geq B_i) &= P(\min_{i=1,\ldots,t} \{A_i - B_i\} \geq 0) \\ &= P(\max_{i=1,\ldots,t} \{-A_i + B_i\} \leq 0). \end{aligned} \tag{3.15}$$

However, similar to (3.2), this method also squeezes many timing quantities into a single canonical form, which incurs large approximation error. The

problem of the linear approximation error has been raised in many statistical timing studies [83, 54, 85], but to compute a joint probability more accurately, all these studies employ Monte Carlo simulation in the end. In an environment with a small number of sources of variation, Monte Carlo simulation can improve the accuracy at a reasonable runtime overhead. However, considered the increasing number of sources of variation in nanometer technologies, the runtime overhead can be excessive. Therefore, we develop an analytic method to compute the joint probability efficiently.

3.3.3 Conditional Probability Computation

We use conditional probabilities to compute the joint probability. So we write

$$\begin{aligned} P(\bigcap_{k=1}^{t} A_k \geq B_k) &= P(A_1 \geq B_1) \cdot P(A_2 \geq B_2 | A_1 \geq B_1) \\ &\quad \cdot P(A_3 \geq B_3 | A_1 \geq B_1, A_2 \geq B_2) \\ &\quad \vdots \\ &\quad \cdot P(A_t \geq B_t | A_1 \geq B_1, A_2 \geq B_2, \cdots, A_{t-1} \geq B_{t-1}). \end{aligned} \tag{3.16}$$

We define an event S_k as

$$S_k \equiv A_1 \geq B_1, A_2 \geq B_2, ..., A_k \geq B_k. \tag{3.17}$$

Then, Eq. (3.16) can be re-written as

$$P(\bigcap_{k=1}^{t} A_k \geq B_k) = \prod_{k=1}^{t} P(A_k \geq B_k | S_{k-1}). \tag{3.18}$$

We also define conditional random variables $A_{c,k}$ and $B_{c,k}$ as

$$A_{c,k} \equiv A_k|S_{k-1} \text{ and } B_{c,k} \equiv B_k|S_{k-1}. \tag{3.19}$$

Then, since

$$\prod_{k=1}^{t} P(A_k \geq B_k|S_{k-1}) = \prod_{k=1}^{t} P(A_{c,k} \geq B_{c,k}), \tag{3.20}$$

we can compute the path criticality by obtaining the conditional random variables.

Let

$$d_{(v_{k-1},v_k)} = f_0^{(k)} + \sum_{i=1}^{n} f_i^{(k)} \Delta X_i + f_{n+1}^{(k)} \Delta R_d^{(k)}. \tag{3.21}$$

Then according to the definition of A_k (3.13), we obtain

$$A_k = g_0 + \sum_{i=1}^{n} g_i \Delta X_i + \sum_{j=1}^{k} f_{n+1}^{(j)} \Delta R_d^{(j)}. \tag{3.22}$$

where $g_0 = \sum_{j=1}^{k} f_0^{(j)}$ and $g_i = \sum_{j=1}^{k} f_i^{(j)}$. We represent the changes of ΔX_i due to given conditions by

$$\begin{aligned}
\Delta Y_i^{(1)} &= \Delta X_i \\
\Delta Y_i^{(2)} &= \Delta X_i | A_1 \geq B_1 \\
\Delta Y_i^{(3)} &= \Delta X_i | A_1 \geq B_1, A_2 \geq B_2 \\
&\vdots \\
\Delta Y_i^{(t)} &= \Delta X_i | A_1 \geq B_1, A_2 \geq B_2, ..., A_{t-1} \geq B_{t-1}.
\end{aligned} \tag{3.23}$$

For random independent components, we define ΔZ_d by

$$\begin{aligned}
\Delta Z_d^{(1)} =& f_{n+1}^{(1)} \Delta R_d^{(1)} \\
\Delta Z_d^{(2)} =& f_{n+1}^{(1)} \Delta R_d^{(1)} | A_1 > B_1 + f_{n+1}^{(2)} \Delta R_d^{(2)} | A_1 \geq B_1 \\
& \vdots \\
\Delta Z_d^{(t)} =& \sum_{j=1}^{t} f_{n+1}^{(j)} (\Delta R_d^{(j)} | A_1 \geq B_1, ..., A_{t-1} \geq B_{t-1}).
\end{aligned} \tag{3.24}$$

Using the conditional random variables, $A_{c,k}$ is represented as

$$A_{c,k} = g_0 + \sum_{i=1}^{n} g_i \Delta Y_i^{(k)} + \Delta Z_d^{(k)}. \tag{3.25}$$

We obtain B_k according to (3.14) and denote it as

$$B_k = h_0 + \sum_{i=1}^{n} h_i \Delta X_i + h_{n+1} \Delta R_b. \tag{3.26}$$

Then, $B_{c,k}$ is represented as

$$B_{c,k} = h_0 + \sum_{i=1}^{n} h_i \Delta Y_i^{(k)} + h_{n+1} \Delta R_b. \tag{3.27}$$

Thus, if we obtain the distributions of $\Delta Y_1^{(k)}, ..., \Delta Y_n^{(k)}$ and $\Delta Z_d^{(k)}$ for $k = 1, ...t$, then we can also get $A_{c,k}$ and $B_{c,k}$ for $k = 1, ..., t$.

From (3.23), we can obtain a recursive formula which is

$$\begin{aligned}
\Delta Y_i^{(k+1)} =& \Delta X_i | A_1 \geq B_1, \cdots, A_k \geq B_k \\
=& \Delta Y_i^{(k)} | A_{c,k} \geq B_{c,k}.
\end{aligned} \tag{3.28}$$

Similarly, from (3.24), we can get another recursive formula which is

$$\begin{aligned}
&\Delta Z_d^{(k+1)} = \sum_{j=1}^{k+1} f_{n+1}^{(j)} (\Delta R_d^{(j)} | A_1 \geq B_1, \cdots, A_k \geq B_k) \\
=& (\Delta Z_d^{(k)} | A_{c,k} \geq B_{c,k}) + f_{n+1}^{(k+1)} (\Delta R_d^{(k+1)} | A_{c,k} \geq B_{c,k}) \\
=& (\Delta Z_d^{(k)} | A_{c,k} \geq B_{c,k}) + f_{n+1}^{(k+1)} \Delta R_d^{(k+1)}.
\end{aligned} \tag{3.29}$$

The last equality follows from the fact that $\Delta R_d^{(k+1)}$ is independent of both A_k and B_k. The following theorem with these two recursive formulas allows us to characterize $\Delta Y_1^{(k+1)}$,...,$\Delta Y_n^{(k+1)}$ and $\Delta Z_d^{(k+1)}$ from $\Delta Y_1^{(k)}$,...,$\Delta Y_n^{(k)}$ and $\Delta Z_d^{(k)}$ when we assume that the kth random variables are normally distributed. This assumption is common in SSTA to re-use the output of an operation as input.

Theorem 3. *Let X, Y, T and U be normally distributed with expected values μ_x, μ_y, μ_t and μ_u, respectively, and with variances σ_x^2, σ_y^2, σ_t^2, and σ_u^2. Then,*

$$E[T|X > Y] = \mu_t + \beta(cov[X,T] - cov[Y,T])/a \tag{3.30}$$

and

$$\begin{aligned} cov[T,U|X > Y] = cov[T,U] - (\beta^2 + \alpha\beta)(cov[X,T] \\ - cov[Y,T])(cov[X,U] - cov[Y,U])/a^2 \end{aligned} \tag{3.31}$$

where

$$a = \sqrt{\sigma_x^2 + \sigma_y^2 - 2cov[X,Y]},$$

$$\alpha = (\mu_x - \mu_y)/a \text{ and } \beta = \varphi(\alpha)/\Phi(\alpha).$$

Proof. The proof is presented in Appendix A. □

Let $(\Delta Y_1^{(k)}$, ..., $\Delta Y_n^{(k)}$,$\Delta Z_d^{(k)})\sim \mathcal{N}(\mathbf{M_k}, \mathbf{C_k})$. To apply this theorem, we need the characteristics of $A_{c,k}$ and $B_{c,k}$ (corresponding to X and Y). If we have $\mathbf{M_k}$ and $\mathbf{C_k}$, we can characterize $A_{c,k}$ and $B_{c,k}$ by

$$E[A_{c,k}] = g_0 + [g_1 \ ... \ g_n \ 1]\mathbf{M_k}, \tag{3.32}$$

$$E[B_{c,k}] = h_0 + [h_1 \ ... \ h_n \ 0]\mathbf{M_k}, \tag{3.33}$$

$$var[A_{c,k}] = [g_1 \ ... \ g_n \ 1]\mathbf{C_k}[g_1 \ ... \ g_n \ 1]^T, \tag{3.34}$$

$$var[B_{c,k}] = [h_1 \ ... \ h_n \ 0]\mathbf{C_k}[h_1 \ ... \ h_n \ 0]^T + h_{n+1}^2, \tag{3.35}$$

$$cov[A_{c,k}, B_{c,k}] = [g_1 \ ... \ g_n \ 1]\mathbf{C_k}[h_1 \ ... \ h_n \ 0]^T. \tag{3.36}$$

Hence, $\mathbf{M_k}$ and $\mathbf{C_k}$ result in $\mathbf{M_{k+1}}$ and $\mathbf{C_{k+1}}$ by (3.30), (3.31) and the two recursive formulas, and starting with $\mathbf{M_1} = \mathbf{0}$ and $\mathbf{C_1} = diag([1 \ ... \ 1 \ (f_{n+1}^{(1)})^2])$, we can obtain $\mathbf{M_k}$ and $\mathbf{C_k}$ for all $k = 2, ..., t$. Finally, substituting (3.32-3.36) into (1.2) yields $P(A_{c,k} \geq B_{c,k})$ for all $k = 1, ..., t$, and we obtain the path criticality by (3.20). If we let $f_{n+1}^{t+1} = 0$, $A_{c,t+1}$ becomes the conditional delay of p given p is the critical path. The conditional distribution can be obtained from (3.32) and (3.34).

3.3.4 Summary

Algorithm 4 summarizes the overall computation procedure. In Algorithm 4, A, B and d are represented in the standard canonical form, and the addition and the maximum are the statistical operations provided by parameterized SSTA (line 4 and 10). The function $SensitivityVector$ returns the sensitivity values to each sources of variation from a timing quantity represented in the canonical form (line 5,13, and 14) The random variable $A_{c,i}$ ($B_{c,i}$) is the conditional random variable of A_i (B_i) given $A_1 \geq B_1, ..., A_{i-1} \geq B_{i-1}$, and their moments are computed by (3.32-3.36) (line 15 and 16). Theorem 3 is applied to every source of variation, and the two equations in line 21 and 22 are the matrix form of (3.30) and (3.31). Note that $\Phi(\alpha)$ is $P(A_{c,i} > B_{c,i})$ (line 19).

Algorithm 4 ComputePathCriticality

Input: $p = (v_0, ..., v_t)$
Output: $\lambda(p)$

1: $A_0 \leftarrow 0$, $\lambda \leftarrow 1$
2: $\mathbf{M_1} \leftarrow \mathbf{0}$, $\mathbf{C'_1} \leftarrow diag([1\ ...\ 1\ 0])$
3: **for** $i = 1, ..., t$ **do**
4: $\quad A_i \leftarrow A_{i-1} + d_{(v_{i-1}, v_i)}$
5: $\quad [f_1, ..., f_n, f_{n+1}] \leftarrow \text{SensitivityVector}(d_{(v_{i-1}, v_i)})$
6: $\quad \mathbf{C_i} \leftarrow \mathbf{C'_i} + diag([0\ ...\ f_{n+1}^2])$
7: $\quad B_i \leftarrow 0$
8: $\quad$ **for** each direct predecessor u of v_i **do**
9: $\quad\quad$ **if** $u \neq v_{i-1}$ **then**
10: $\quad\quad\quad B_i \leftarrow \max\{B_i, at(u) + d(u, v_i)\}$
11: $\quad\quad$ **end if**
12: $\quad$ **end for**
13: $\quad [g_1, ..., g_n, g_{n+1}] \leftarrow \text{SensitivityVector}(A_i)$
14: $\quad [h_1, ..., h_n, h_{n+1}] \leftarrow \text{SensitivityVector}(B_i)$
15: $\quad$ compute $E[A_{c,i}], var[A_{c,i}]$
16: $\quad$ compute $E[B_{c,i}], var[B_{c,i}], cov[A_{c,i}, B_{c,i}]$
17: $\quad a \leftarrow \sqrt{var[A_{c,i}] + var[B_{c,i}] - 2cov[A_{c,i}, B_{c,i}]}$
18: $\quad \alpha \leftarrow (E[A_{c,i}] - E[B_{c,i}])/a$, $\beta \leftarrow \varphi(\alpha)/\Phi(\alpha)$
19: $\quad \lambda \leftarrow \lambda \times \Phi(\alpha)$
20: $\quad \mathbf{D} \leftarrow \mathbf{C_i}([g_1\ ...\ g_n\ 1] - [h_1\ ...\ h_n\ 0])^T$
21: $\quad \mathbf{M_{i+1}} \leftarrow \mathbf{M_i} + \beta\mathbf{D}/a$
22: $\quad \mathbf{C'_{i+1}} \leftarrow \mathbf{C_i} - (\beta^2 + \alpha\beta)\mathbf{D}\mathbf{D}^T/a^2$
23: **end for**
24: **return** λ

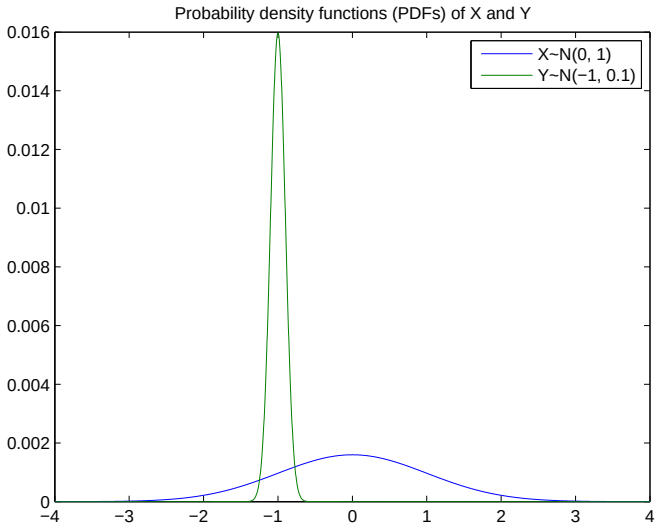
While Algorithm 4 is specially optimized for the path criticality computation, we can easily modify it for the general purpose of computing a joint probability of normal random variables; Algorithm 4 implicitly involves the cumulative distribution function for multivariate normal random variables represented in the canonical form (i.e. multidimensional Q-function). Thus, this technique can be employed in many other applications of statistics timing analysis.

3.4 Approximation Error Analysis

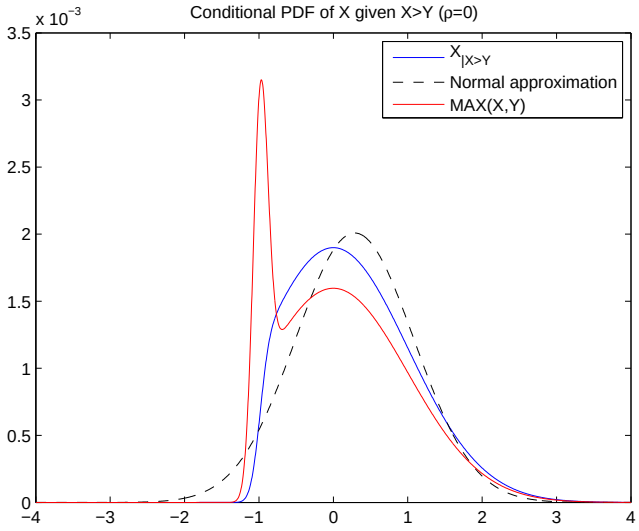
In the previous section, we have assumed that the conditional probability density functions (PDFs) follow normal distributions which incurs some approximation error. The authors in [68] formally analyzed the approximation error in the max operation, and we will perform the same analysis for our conditioning operation, comparing the accuracy of the two operations. The probability density function of a random variable is denoted by ψ. For a random variable X approximating to a random variable Y, we quantify the error by

$$\Xi_{(X)(Y)} \equiv \int_{-\infty}^{\infty} |\psi_X(t) - \psi_Y(t)| dt. \tag{3.37}$$

Let X and Y be normally distributed random variables where $X \sim \mathcal{N}(\mu_x, \sigma_x)$, $Y \sim \mathcal{N}(\mu_y, \sigma_y)$. Their correlation coefficient is denoted by ρ. Let $W = X|X > Y$ and we denote its normal approximation by W_G. Using the derivations



(a) The PDFs of X and Y



(b) The PDFs of $X|X > Y$ and $MAX(X, Y)$

Figure 3.5: In both the maximum operation and our proposed conditioning operation, the normal approximation is inevitable for efficiency. However, the approximation error is much less in the conditioning operation.

in [68], we can easily obtain

$$\psi_W(t) = \frac{1}{\sigma_X}\phi\left(\frac{t-\mu_x}{\sigma_x}\right)\Phi\left[\frac{\left(\frac{t-\mu_Y}{\sigma_Y}\right)-\rho\left(\frac{t-\mu_X}{\sigma_X}\right)}{(1-\rho^2)^{1/2}}\right]. \tag{3.38}$$

The derivative of (3.38) implies that the conditional PDF of X given $X > Y$ is *unimodal* in contrast to the PDF of $\max(X, Y)$. Figure 3.5(a) shows the PDFs of X and Y when $\mu_x = 0, \sigma_x = 1, \mu_y = -1, \sigma_y = 0.1$ and $\rho = 0$, and Figure 3.5(b) shows the conditional PDF of X given $X > Y$. The conditional PDF has almost the same shape as the PDF of X in the region of $X > 0$ because the term Φ in (3.38) becomes one in that region. However, in the region of $X < 0$, the value of term Φ sharply drops from one to zero which determines the shape of the conditional PDF in that region. In Figure 3.5(b), we superpose the normal approximation to the conditional PDF, and readers can identify that the conditional PDF has better similarity to its normal approximation compared to the PDF of $\max(X, Y)$. Note that $\Xi_{(W)(W_G)} = 0.2$ in this example.

The approximation error is a function of $\mu_x, \sigma_x, \mu_y, \sigma_y$ and ρ, and for comprehensive analysis, we may need to examine all combinations of these parameters. However, defining

$$a \equiv \sqrt{\sigma_x^2 + \sigma_y^2 - 2\sigma_x\sigma_y\rho} \tag{3.39}$$

$$\alpha \equiv (\mu_x - \mu_y)/a \tag{3.40}$$

and applying the same procedure as [68] for the conditioning operation allow us to fix $\mu_x = 0$ and $\sigma_y = 1$ and just to examine combinations of α, σ_y and ρ

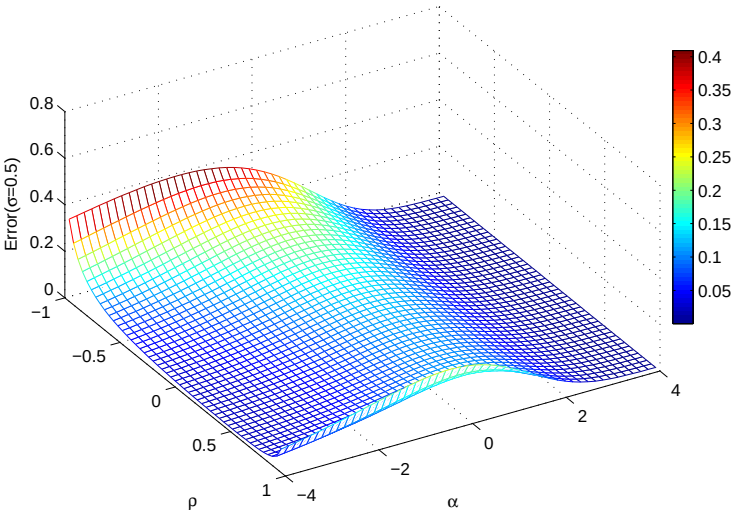


Figure 3.6: $\Xi_{(W)(W_G)}$ as a function of ρ and α when $\sigma_Y = 0.4$

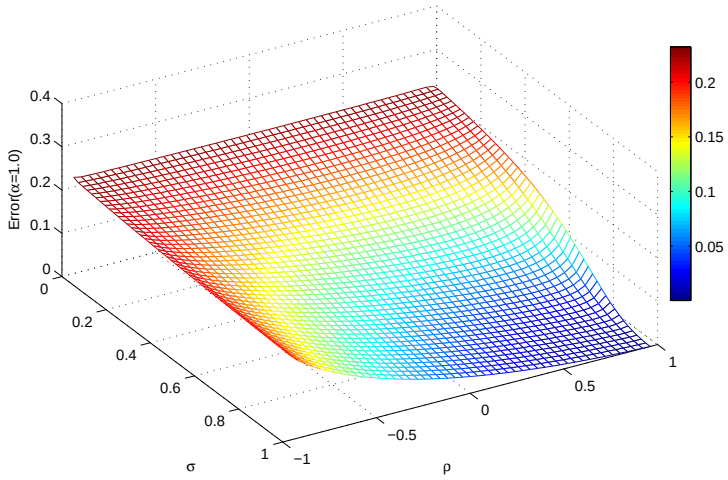


Figure 3.7: $\Xi_{(W)(W_G)}$ as a function of σ_Y and ρ when $\alpha = 1$

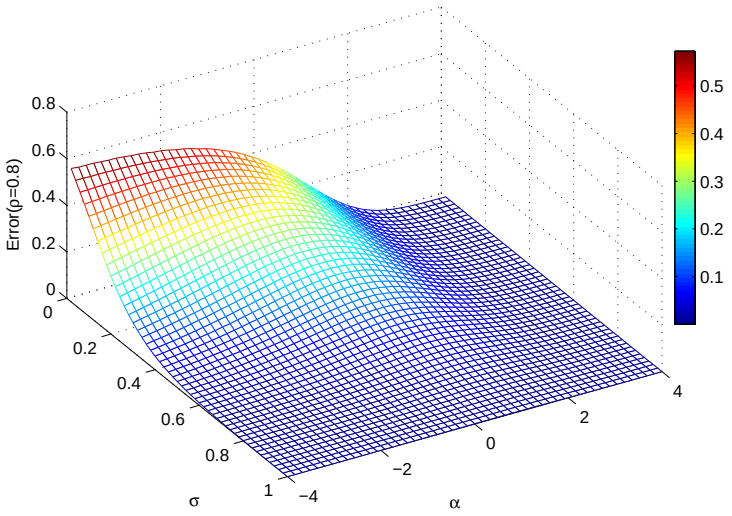


Figure 3.8: $\Xi_{(W)(W_G)}$ as a function of σ_Y and α when $\rho = 0.8$

in the region of

$$\begin{aligned} -4 \leq \quad & \alpha \quad \leq 4, \\ 0 \leq \quad & \sigma_y \quad \leq 1, \\ -1 \leq \quad & \rho \quad \leq 1. \end{aligned} \tag{3.41}$$

Figure 3.6, 3.7 and 3.8 show the approximation error as a function of two parameters when the other parameter is fixed. Since the same fixed points as [68] are used, these figures can be directly compared with those in [68]. The comparison reveals that the approximation error in the conditioning operation is significantly less than that in the max operation. In particular, the error is decreasing as ρ increases and becomes very small when X and Y are positively correlated as shown in Figure 3.7 and 3.8. Therefore, our conditioning operation fits well in statistical timing analysis where most random variables

are positively correlated.

3.5 Experimental Results

3.5.1 Conditioning Operation

We first demonstrate the accuracy and runtime of the proposed conditioning operation in computing a joint probability, comparing it to the max operation and 6 different Monte Carlo simulations with different numbers of samples. We have implemented all these methods in C++, and a Basic Linear Algebra Subprograms (BLAS) package has been used for vector and matrix operations. This comparison has been performed on a 3.0GHz 2 Xeon X5570 Linux machine. For this experiment, we randomly generate 10 timing quantities represented in the canonical form with 21 global sources of variation. The mean value of each timing quantity is randomly selected within a range of [1.0, 3.0] and the standard deviation is also randomly chosen from 10% to 20% of the mean value. Given the 10 timing quantities, each method computes the probability that one timing quantity selected randomly is greater than the other timing quantities. For the max operation, this probability is computed by using (3.15). We run Monte Carlo simulation with 100,000 samples for golden values and obtain the absolute error for each method. We perform this experiment in two different cases. In the first case, the sensitivity values to each sources of variation are randomly generated within a range of [-1.0, 1.0], and the average correlation coefficient between the selected timing quantity and the other 9 timing quantities becomes 0.1. In the second case, the

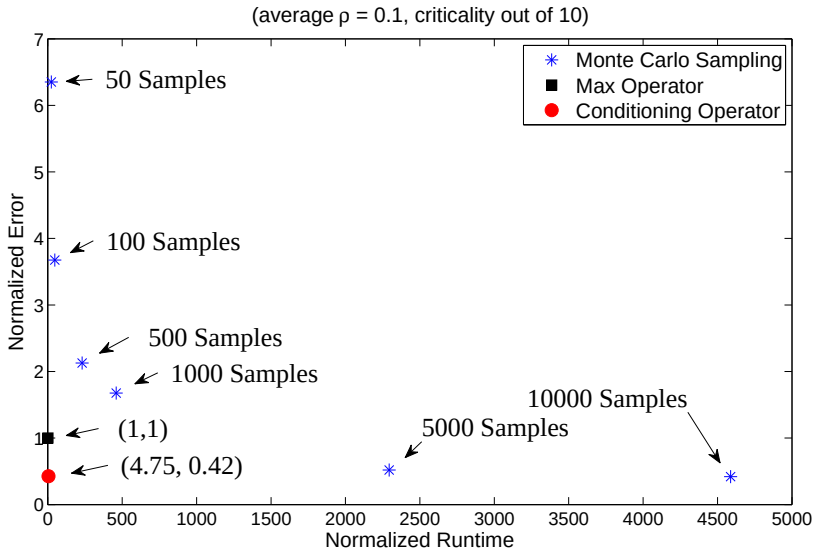


Figure 3.9: Each method computes a joint probability in a weakly correlated environment.

sensitivity values are within a range of [0, 1.0], and the average correlation coefficient becomes 0.82. Figure 3.9 and 3.10 shows the absolute error and the runtime of each method in the two cases, respectively. The runtime and the absolute error of each method are normalized to those of the max operation. In Figure 3.9 and 3.10, the results of the six Monte Carlo simulations exhibit the tradeoff of the runtime and the accuracy. Clearly, the max operation and the conditioning operation are below the tradeoff curve, and they are Pareto improvements to some Monte Carlo simulations. In the first case, the conditioning operation provides 2.38X accuracy improvement over the max operation at the cost of 4.75X runtime overhead as shown in Figure 3.9. The Monte Carlo simulation with 10^4 samples requires 4500X runtime overhead

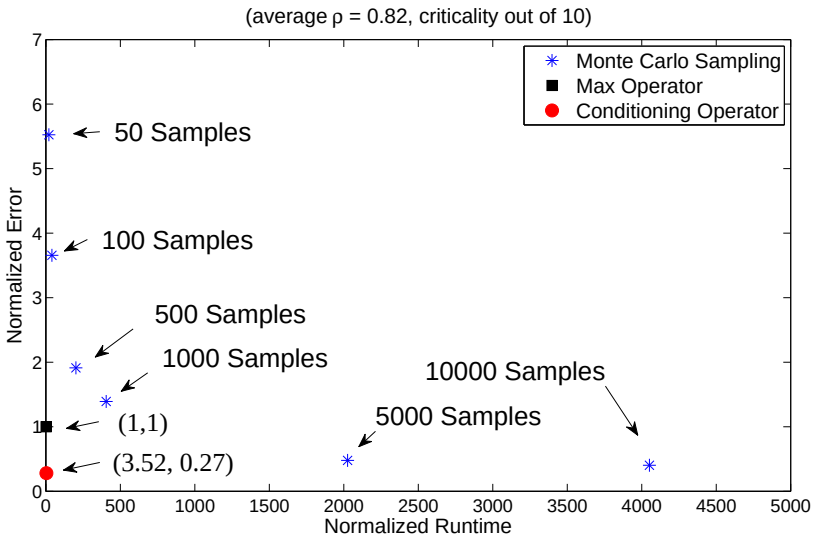


Figure 3.10: Each method computes a joint probability in a strongly correlated environment.

to obtain similar accuracy to the conditioning operation. In the second case, the conditioning operation achieves significantly better accuracy as expected in Section 3.4. In this case, the runtime is also improved because if a given condition is true in the whole sample space, the conditional statistics do not need to be updated, and we encounter such a case more often.

In this experiment, we have computed the criticality probability of one timing quantity among 10 timing quantities. If we need to compute the criticality probabilities of all the 10 timing quantities, the conditioning operation need to repeat the same procedure 10 times, while Monte Carlo simulations produce them at the same time. In the max operation, this is also possible if the *binary partition tree* proposed in [76] is employed. On the other hand, if the criticality probability of only one timing quantity is needed as in this experi-

ment, and we need to obtain the criticality *incrementally* (i.e. we find the other 9 timing quantity gradually and need to maintain the criticality of the timing quantity among the timing quantities found so far), the improvements of the conditioning operation becomes even bigger than that in this experiment. We often encounter this scenario in statistical path selection applications (e.g., the selection of paths whose criticality is greater than a threshold value). Thus, the improvements can vary depending on applications.

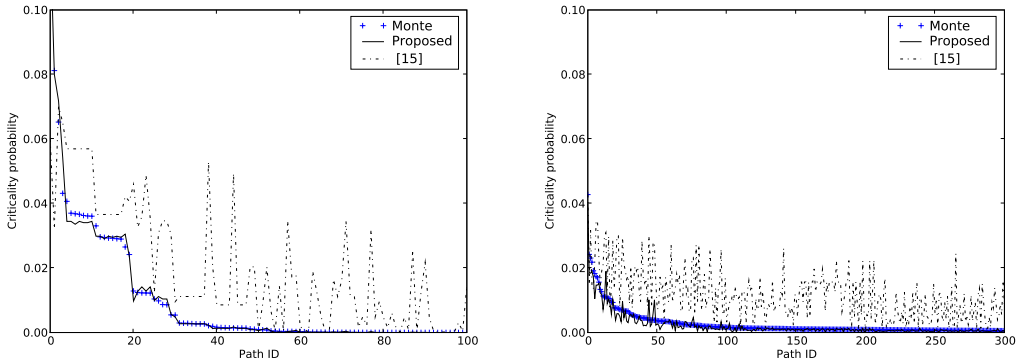
3.5.2 Criticality Computation

We have implemented the proposed algorithm in C++, and have used ISCAS85 circuits as benchmarks. We also have used the or1200 open-source microprocessor including a 32-bit, 5-stage Wallace multiplier and s9234 in ISCAS89 since it contains many near-critical paths. All benchmarks are technology-mapped to the TSMC 180nm standard cell library, and the nominal delay annotated to each edge in the timing graph is the sum of the gate and wire delays obtained from the Standard Delay Format (SDF) file.

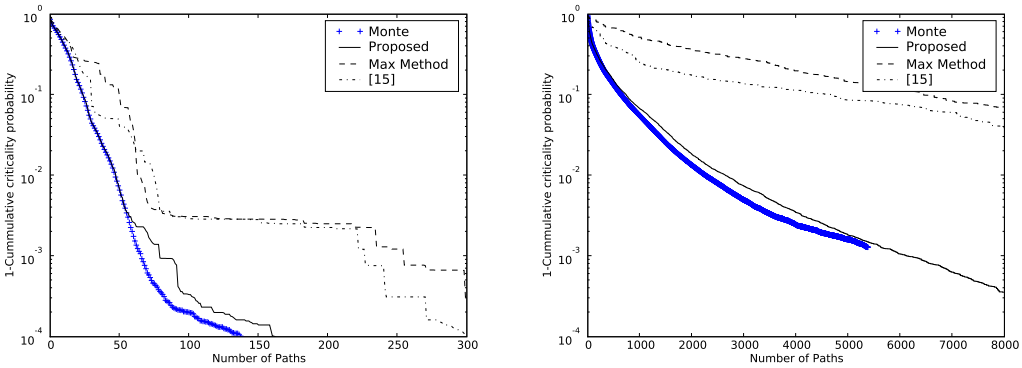
In this experiment, a quad tree with 3 levels is used to model spatial correlation which results in 21 global sources of variation. This allows us to verify the proposed algorithm under difficult conditions. The global sources of variation associated with the first, the second and the third level can change the delay of each edge up to $\pm 4\%$, $\pm 5\%$ and $\pm 6\%$ of its nominal value in the 3σ case, respectively. The delay of each edge also has the independent variation, and the 3σ point is 5% of the nominal delay. Our statistical static

timing tool uses the SSTA algorithm proposed in [15]. In the timing tool, we have implemented the second method (excluding the given path) proposed in [77] for comparison. To demonstrate the accuracy of the conditioning operation, we also calculate the path criticality by (3.15). This is called the *max method*. Note that the proposed method and the max method compute the path criticality from the same timing quantities simplified by the algebraic elimination.

Figure 3.11 compares the accuracy of each method in the circuits s9234 and or1200. We first select top 100 (300) paths with largest criticality probability using Monte Carlo simulation from s9234 (or1200) and then compute the criticality probabilities of the paths by each method. Figure 3.11(a) shows the criticality probabilities. Unlike [77], the results of our proposed method are very close to the Monte Carlo simulation results. Figure 3.11(b) shows the cumulative criticality probabilities computed by Monte Carlo simulation for up to top 300 (8000) paths of s9234 (or1200) with largest criticality probability in each method. In other words, each method ranks paths differently, and for each list of paths, we compute the cumulative criticality by Monte Carlo simulation. Thus, the values shown in Figure 3.11(b) are accurate, and Figure 3.11(b) shows how well each list is ranked. The proposed method outperforms the two other methods. So as not to clutter the plot, the max method is not shown in Figure 3.11(a), but its poor accuracy is implied by Figure 3.11(b). The circuit or1200 is highly optimized and has numerous near-critical paths which make the max method and the method of [77] more inaccurate. In this situation,



(a) criticality probability in s9234 and or1200



(b) 1-cummulative criticality probability in s9234 and or1200

Figure 3.11: The proposed method not only shows significant better accuracy as compared to the existing method but also is comparable to Monte Carlo simulation.

Table 3.1: Accuracy comparison

		[77]		Max Method		Proposed	
	#p	max.ε	avg.ε	max.ε	avg.ε	max.ε	avg.ε
c17	1	0.0003	0.0003	0.0004	0.0004	0.0002	0.0002
c432	6	0.0816	0.0380	0.0636	0.0257	0.0016	0.0006
c499	37	0.0247	0.0185	0.0696	0.0494	0.0015	0.0006
c880	4	0.1399	0.0761	0.0649	0.0277	0.0074	0.0032
c1355	23	0.0359	0.0272	0.0766	0.0654	0.0015	0.0004
c1908	2	0.2157	0.1522	0.0058	0.0042	0.0006	0.0004
c2670	8	0.1985	0.0504	0.1496	0.0374	0.0815	0.0250
c3540	4	0.2168	0.0740	0.2272	0.0665	0.0463	0.0213
c5315	4	0.1075	0.0619	0.0637	0.0463	0.0239	0.0159
c6288	3	0.5486	0.2033	0.4838	0.1697	0.3252	0.1191
c7552	1	0.0896	0.0896	0.0002	0.0002	0.0008	0.0008
avg.		0.1508	0.0720	0.1096	0.0448	0.0446	0.0170

the improvement of our method becomes more significant.

In order to compare the accuracy of each method in ISCAS85 benchmark circuits, we select the paths whose criticality probabilities are greater than 0.01 by Monte Carlo simulation, and obtain the criticality values for them from each method. Table 3.1 shows the average absolute error and the maximum error of each method. The first column in the table shows the number of the selected paths. The max method shows better accuracy on average than the method of [77] since it uses the inequalities simplified by the algebraic elimination. The single canonical form used in both the max method and the method of [77] can be accurate if the design have only one potential critical path (a very long path). In this case, the maximum of all the path delays in the design always equals the critical path delay, which means that the max-

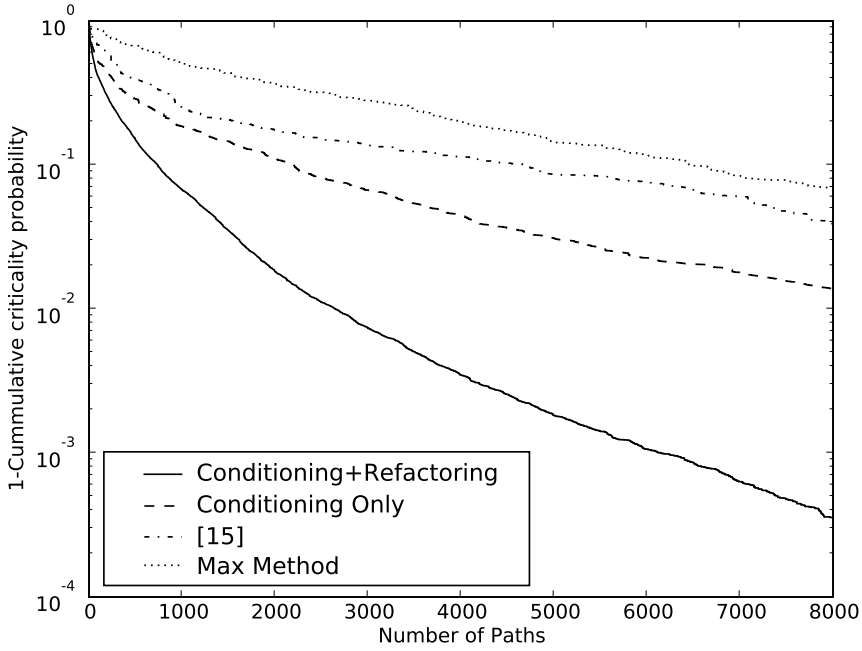


Figure 3.12: Both refactoring and the conditioning operation are crucial to achieve the accuracy close to that of Monte Carlo simulations.

imum is still linear and the linear approximation to the maximum does not cause error. The circuits c17 and c7552 are close to this case, and the max method and the method of [77] shows pretty accurate results for them. On the other hand, the circuits c499 and c1355 are several near-critical paths, and both the methods suffer from large error. Especially, the formulation used in the max method is more susceptible to such a case. Unlike the two methods, the proposed method provides accurate results even for c499 and c1355. Since industrial designs have a number of near-critical paths, the benefit of the proposed method can be much bigger in industrial designs.

Table 3.2: Edge criticality computation results

	[76]	Conditioning Only		Cond.+Refactoring	
	avg.ε	avg.ε	norm.ε	avg.ε	norm.ε
c17	0.00008	0.00007	0.932	0.00007	0.932
c432	0.00510	0.00228	0.447	0.00010	0.020
c499	0.00328	0.00126	0.385	0.00052	0.159
c880	0.00150	0.00019	0.127	0.00019	0.127
c1355	0.00192	0.00171	0.890	0.00040	0.206
c1908	0.00260	0.00181	0.695	0.00004	0.014
c2670	0.00147	0.00108	0.733	0.00035	0.237
c3540	0.00151	0.00172	1.138	0.00052	0.342
c5315	0.00102	0.00047	0.460	0.00028	0.271
c6288	0.00109	0.00056	0.517	0.00025	0.232
c7552	0.00035	0.00036	1.039	0.00000	0.010
eth	0.00022	0.00007	0.322	0.00003	0.154
tv80	0.00062	0.00034	0.544	0.00021	0.332
or1200	0.00083	0.00020	0.237	0.00012	0.141
Avg.	0.00154	0.00087	0.605	0.00022	0.227

3.5.3 Breakdown of the Accuracy Improvement

The accuracy improvement of the proposed method is achieved by the combination of the conditioning operation and the refactoring technique. To show the contribution of each technique, we rank paths in or1200 again using the proposed method with and without refactoring, and the results are shown in Figure 3.12. Significant part of the improvement comes from the refactoring technique. However, it is important to note that without the conditioning operation, the accuracy cannot be improved at all. In Figure 3.12, the result of the max method is obtained using refactoring, but due to the inaccuracy of the max operator, it cannot take advantage of refactoring.

We have not presented the details of the edge criticality computation. However, we can also take advantage of refactoring and the conditioning operation there. We compare our method with the conventional cutset-based method [76]. We compute the criticality of each edge using each method and obtain the golden values from Monte Carlo simulations with 40,000 samples. For each edge, the error is computed, and the average error for each circuit is calculated. Also, we normalize the average error to that of the cutset-based method. Table 3.2 shows the average errors and the normalized errors. If only the conditioning operation is used, the error is reduced by 40% on average, although there are some glitches in c3540 and c7552. However, if our two techniques are combined, we can achieve a significant error reduction for all benchmark circuits.

3.6 Conclusions

We have proposed a path criticality computation method which significantly improves the accuracy of the path criticality computation compared to the state-of-the-art method, in particular when the circuit has many near-critical paths. Since the conditioning operation used in this chapter is not limited to this specific application, we believe that this operation, combined with the total probability theorem, could replace the max operator where higher accuracy is required. The run time of the proposed method may be a concern for very large industrial circuits. However, the matrix operations can be easily parallelized, and our method can be implemented very efficiently

in today's computing environments with SIMD instructions and multi-core multi-threaded CPUs and GPUs.

Chapter 4

A Concurrent Path Selection Algorithm in Statistical Timing Analysis

4.1 Introduction

The increasing performance of devices, combined with increasing variability in the nanometer regime, has posed new challenges in design and testing methodologies. Circuit timing is subject to factors including process parameters, manufacturing defects, power supply noise, crosstalk and multiple input switching (MIS). Some of these factors are difficult to control precisely during manufacture and others change during operation or within die. Circuit delays thus are uncertain at the design stage. While the variability continues to increase as device technology scales, the margin to deal with the variability remains the same or decreases due to the demand for high performance circuits [56]. Therefore, testing methodologies need to be improved in order to maintain the reliability of the circuits.

In order to deal with various timing defects, several delay fault models have been used such as the transition fault model and the path delay fault model. The transition fault model well captures the behaviors of defects that cause a localized and large change of delay, while the path delay fault model

is suitable for distributed and subtle changes of delay such as process variation delay faults. Under the path delay fault model, the number of faults is enormous and it is common to select some paths to be tested depending on the behaviors of timing defects being targeted. Some path selection methods [41, 50, 63] target localized but small delay defects and others [48, 36] target distributed and small delay defects. Actually, the transition fault model, which aims at large and gross delay defects, can be considered as to guide us to select any path passing through each wire.

Certainly path selection is an important step for delay testing, and considering this problem in statistical domain provides a unified view to all these approaches. In delay testing, our ultimate objective is to test at least one faulty path of each given bad chip. The first-order factor we need to consider is the fault probability of each path in the circuit. Clearly, paths with high faulty probability should be tested. The second-order factor is statistical correlation of the delay variations. If some two paths with high fault probability become faulty together in every produced chip, it is wasteful to test both the two paths.

Traditional path selection algorithms bear in mind the statistical aspects of the delay variations to improve test effectiveness. Since a simple and effective approach is to select paths with high fault probability, it is common to target top k longest paths. Since a spot defect causes many paths to be faulty that pass through the defected site, we can assume strong structural correlation if spot defects are targeted. Then, it is effective to select a longest

path passing through every wire [41, 50, 63]. If spatial correlation is considered, an effective method is to select a few long paths in each block [36]. These conventional methods use the statistical aspects of the delay variations in an ad-hoc manner, while recent statistical path selection algorithms quantify the fault probability and the correlation using statistical timing analysis. The algorithm proposed in [72] iteratively selects a path that detects the most number of bad chips not covered by already selected paths. In other words, it selects paths with high conditional fault probability. However, this conditional fault probability is difficult to be computed in SSTA so it requires multiple Monte Carlo simulations. Recently, the authors in [89] propose two statistical path selection algorithms that can work with SSTA. The first algorithm called BnB-SPM selects top k paths with highest fault probability very efficiently by pruning. However, this does not utilize the correlation. Thus, the authors define a test quality metric that accounts for the correlation and propose the second algorithm called BnB-JPM, which in a greedy manner, selects paths that maximize the test quality metric.

Basically, these statistical algorithms do not target specific timing defects, and the target defects are determined by the statistical timing model used. Ideally, if the statistical timing model captures process variation [71], spot defects, crosstalk [66], MIS [3], power droop [23], aging effects, or any combinations of theses [65], they target the corresponding defects accordingly. However, these extensive applications of the algorithms are limited by several factors in practice. First, timing models are abstracted and simplified

so are inaccurate to some extent. In particular, second-order effects such as crosstalk, MIS and power drop are difficult to be modeled accurately without input vectors. Secondly, under static timing models, the delays of all input vectors sensitizing a single given path are the same, so path selection matters and the path selection problem is separated from automatic test pattern generation (ATPG). However, in order to target the defects due to the second-order effects, path selection and ATPG cannot be dealt separately because the delay of a given path depends largely on input vectors under the effects. Thirdly, analysis algorithms running on the timing models may not be efficient enough. Monte Carlo simulations can account for various timing defects such as spot defects [72] but are too slow to be used in path selection. Delay tests are typically used to screen timing defects in the manufacturing process of ASICs [30] and for speed-binning of high-performance integrated circuits [18]. Also, they are useful in finding speed-limiting paths in the post-silicon debug (a.k.a., post-silicon validation), which is performed to improve the performance and yield of the next silicon iteration (or, spin) of a high-speed microprocessor design [55, 13]. Smart path selection is beneficial in both the manufacturing test and the post-silicon validation, but the above-mentioned factors are more critical in the post-silicon validation where it is necessary to find speed-limiting paths and vectors [55]. In contrast, in the manufacturing test, the correlation between the result of delay tests and the actual operating frequency is more important than finding timing critical paths. Also, since design bugs are already fixed, process variation and spot defects are major

defect mechanisms although other effects may be combined. Therefore, the above mentioned factors are less crucial in high-volume manufacturing test.

Considering current statistical timing modeling techniques in terms of accuracy and efficiency, this chapter primarily addresses process variation delay faults in the manufacturing test of ASICs as in [89]. However, our proposed algorithm is not limited to this specific type of faults and can extend its applications as timing models advance. The major contributions of this chapter include the following:

- Existing statistical algorithms [72, 89] select one path at a time in a greedy manner, whereas our proposed method selects k paths gradually at the same time with a global view. To our best knowledge, this is a new class of statistical path selection.

- The methods in [44, 72] require a Monte Carlo simulation for each path or each defect, while our proposed algorithm needs statistical timing analysis at the beginning of the process only once for k-path selection, and both Monte Carlo sampling and block-based SSTA can be used.

- Existing statistical path selection studies do not consider the issue of untestable paths and validate the quality of produced path sets assuming all paths are testable. By integrating a SAT solver, our algorithm can easily exclude untestable paths from consideration. In addition, we present the post-ATPG quality of produced path sets tackling various issues associated with untestable paths and false paths.

The rest of this chapter is organized as follows. Section 4.2 presents the necessary background for statistical path selection. Section 4.3 discusses the path selection problem in detail. Section 4.4 formulates the concurrent path selection problem and proposes our basic approach, and Section 4.5 extends the basic approach to handle several remaining issues. Section 4.6 presents experimental results, and Section 4.7 concludes this chapter.

4.2 Background

In this section, we first define basic terms used throughout this chapter. A timing graph is an edge-weighted directed acyclic graph (dag). The edge delay (weight) is the sum of the gate delay and the interconnect delay associated with the edge. The arrival time and the delay are denoted by at and d, respectively. The set of all paths in the given circuit is denoted by Ω. The set of all paths passing through a subpath π is denoted by Ω_π. The operating clock period is denoted by T.

4.3 Problem Formulation

The objective of this chapter is to construct a high quality path set using a statistical timing framework. In order to measure the quality of a path test, we will define a quality metric. In the stuck-at fault model, the *fault coverage* is a universally used quality metric and due to its simplicity, the metric was adopted in the path delay fault model. However, the use of fault coverage makes the path delay fault model intractable and ineffective. To

demonstrate why the fault coverage is a misleading metric in the path delay fault model unlike in the stuck-at fault model, we will consider an example.

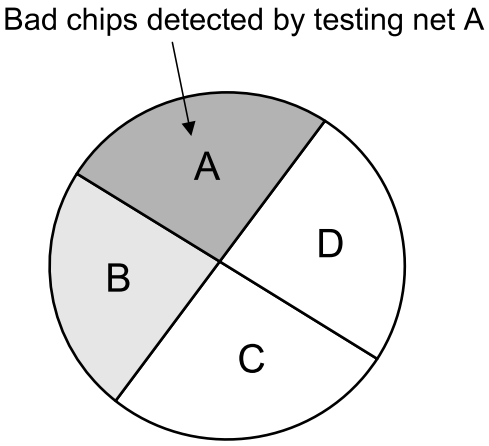


Figure 4.1: The sample space of bad chips with stuck-at faults: We suppose that a produced bad chip corresponds to one element in the sample space so each element is equally likely. The 4 events are not disjoint in practice, but since the intersections are small enough, this figure is acceptable.

Suppose that a design with 4 nets is manufactured. Figure 4.1 shows the sample space of the bad chips with stuck-at faults. Each quarter circle represents bad chips that can be detected by a test. Clearly, our objective is to cover the entire area by some tests; we want to detect all bad chips. However, if the number of tests we can run is limited, our objective is to cover as large area as possible. Usually the probability that each net is defective is not very different, so each test is depicted to take similar areas. If we are allowed to select two nets to be tested out of the four nets, any combination will detect a similar number of bad chips. Thus, we do not need to select nets to be tested,

and the number of tested nets is proportional to the number of detected bad chips; the stuck-at fault coverage is effective. Besides, since it is affordable to test all nets in the circuit, we just try to test all nets. Figure 4.2(a) and (b)

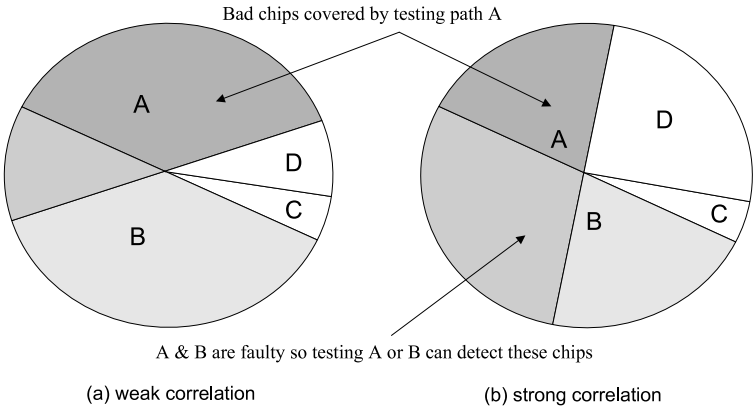


Figure 4.2: Sample space of bad chips with path delay faults

show the sample space of bad chips with path delay faults. Unlike the stuck-at fault model, the fault probability of each path is very different, and the area taken by each test is depicted accordingly. In this case, selection of paths to be tested can yield a substantial difference in test effectiveness. In the path delay fault model, we can consider two different scenarios on the intersections of the tests. Suppose that the areas of the intersections are very small as in Figure 4.2(a). Then, we can just test top k paths with the largest area in the sample space when k paths are affordable, and this can lead to an optimal solution. In this example, if $k = 2$, it is optimal to select A and B. Thus, path selection is an easy task in this scenario. Now suppose that the areas of the intersections are large as shown in Figure 4.2(b). In this scenario, the simple

heuristic as in the first scenario does not result in a good solution. In this case, the paths A and D are better than the paths A and B. To obtain an optimal path set, we may need to inspect every combination. Thus, path selection becomes a very difficult task. This kind of selection problem is known as the *maximum coverage* problem, which is a well-known NP complete problem.

Path delays in the circuit are correlated because of several different mechanisms, which makes the path selection problem similar to the second scenario. If two paths pass through one or more same gates, the two path delays are correlated even if the delays of all gates and interconnects are independent. This is called *structural correlation*. If two gates are closely placed in a chip, they are fabricated or operated under a similar environmental condition, and their delays are increased or decreased together. This is called *spatial correlation*. Also, a source of variation can affect several gates and interconnect delays simultaneously, inducing *global correlation*. Thus, we need to solve the difficult combinatorial optimization problem.

For the maximum coverage problem, it is known that integer linear programming provides an exact solution. Also, there are many heuristics. However, we are required to solve the maximum coverage problem in the probability space. In this case, to solve the problem using integer linear programming, it is necessary to enumerate all paths explicitly and the joint fault probabilities of their combinations should be calculated. Clearly, this is prohibitive and we look for an alternative.

4.3.1 Test Quality Metric

In this subsection, we formally write the concepts discussed above. The *detection probability* of a test Π is defined as

$$q(\Pi) = P(Test\_fail|Chip\_is\_bad). \tag{4.1}$$

It is obvious that the detection probability is the ultimate test quality metric. However, instead of this probability, the stuck-at fault coverage is used universally in the stuck-at fault model which can be justified from the following derivation. Let p be the probability that a net has a stuck-at fault. Also let N be the total number of stuck-at faults in the circuit and let c be the stuck-at fault coverage of Π. Assuming that the occurrences of stuck-at faults are independent events, we obtain

$$q(\Pi) = \frac{1-(1-p)^{Nc}}{1-(1-p)^{N}}. \tag{4.2}$$

If p is sufficiently small, we can assume that multiple stuck-at faults do not occur. Then,

$$q(\Pi) = c. \tag{4.3}$$

In the path delay fault model, the above assumptions are not reasonable, so we need to go back to the ultimate test quality metric to evaluate the quality of delay tests accurately. Let us ignore the testability issue of paths for a moment and assume that all paths are robustly testable. This issue will be handled later. If a static timing model is employed, we can easily generate test patterns sensitizing given paths and conversely given test patterns, it

only matters which paths are sensitized. Thus we can say that a path set is equivalent to a test suite, and Π denote a path set. Thus we can write

$$\begin{aligned} q(\Pi) &= P(Test\_fail|Chip\_is\_bad) \\ &= P\left(\bigcup_{p\in\Pi} d(p) > T | \bigcup_{p\in\Omega} d(p) > T\right) \\ &= P\left(\max_{p\in\Pi}\{d(p)\} > T | \max_{p\in\Omega}\{d(p)\} > T\right) \\ &= \frac{P(\max_{p\in\Pi}\{d(p)\} > T)}{P(\max_{p\in\Omega}\{d(p)\} > T)}. \end{aligned} \tag{4.4}$$

In deterministic timing, this probability cannot be calculated, but recent SSTA can compute it very efficiently, allowing us to use the ultimate quality metric directly. Since our work mainly targets process variation delay faults by employing an adequate timing model, the sample space of all produced chips becomes the process (parameter) space and we will call the detection probability *process space coverage metric* (PCM) as in [89].

We mathematically formulate the path selection problem for delay fault testing as follows:

$$\begin{aligned} \max_{\Pi\subset\Omega} \quad & q(\Pi) \\ \text{s.t.} \quad & |\Pi| = k. \end{aligned} \tag{4.5}$$

Since the denominator of the detection probability is not a function of Π, we can eliminate the denominator in our objective function. The problem then becomes

$$\begin{aligned} \max_{\Pi\subset\Omega} \quad & P(\bigcup_{p\in\Pi} d(p) > T) \\ \text{s.t.} \quad & |\Pi| = k. \end{aligned} \tag{4.6}$$

4.4 Proposed Method

Using the randomness in the manufacturing process of chips, we can consider a *gamble*. Let a gambler play the gamble. We partition the set of all paths in a given design into two equally sized groups. The gambler is supposed to bet his entire capital on this game, dividing it into two parts for each group. Let us take a manufactured chip. Suppose that the manufactured chip has only one faulty path. If there exists a faulty path in one group, the gambler will get his bet on the group back as a payoff (i.e., the odds are one). Otherwise, he loses his bet. The probability of winning for each group is given to the gambler before betting from the design. If he has the remaining capital, we can partition the group with the faulty path into two groups again, and the gambler can play another game. If there are N paths in the design, he can play up to $logN$ games. If he wins in all these games, he actually succeeds to predict the faulty path from the design. In this scenario, the gambler was able to proceed only if he wins. Let us consider another scenario. Suppose that his initial capital is k and a bet must be an integer. He plays the first game. Without letting the gambler know the outcome, he can simulate playing the next game for each possible outcome. This process can be continued which is equivalent to drawing a *decision tree*. At the end of this simulation process, each group contains only one path and some paths will get the gambler's bet. At most k paths can get some bets and others will get nothing. A smart gambler selects good candidate paths from the design.

4.4.1 Partitioning Path Sets

We take a design as input and construct a timing graph. For simplicity, we assume that rising and falling delays are the same. Then, the timing graph is an edge-weighted, directed acyclic graph (dag). However, if the technique suggested in [15] is employed, we can construct the same type of a graph that considers both rising and falling delays and has twice the number of paths for falling and rising cases as that in the original timing graph. Thus, without loss of generality, we can consider the timing graph to be a simple edge-weighted dag.

We perform statistical static timing analysis on the timing graph and can obtain arrival times, required arrival times and slacks for each vertex. We add a vertex called the *source*. The *start points* (i.e., the vertices without incoming edges) corresponds to primary inputs and outputs of flops. Each start point is connected to the source by the edge whose delay is the arrival time of the start point. We add a vertex called the *sink*. The *end points* (i.e., the vertices without outgoing edges) correspond to primary outputs and inputs of flops. Each end point is connected to the sink by the edge whose delay is the required arrival time of the end point. This modified timing graph is called the *augmented timing graph* and the construction process from a design is well illustrated in [77]. We denote the source and the sink by s and t, respectively. In the augmented timing graph, all paths start with s and end at t. From the sink t in the augmented timing graph, we perform the recursive depth-first search which implicitly constructs g a depth-first search tree. Figure 4.3 shows

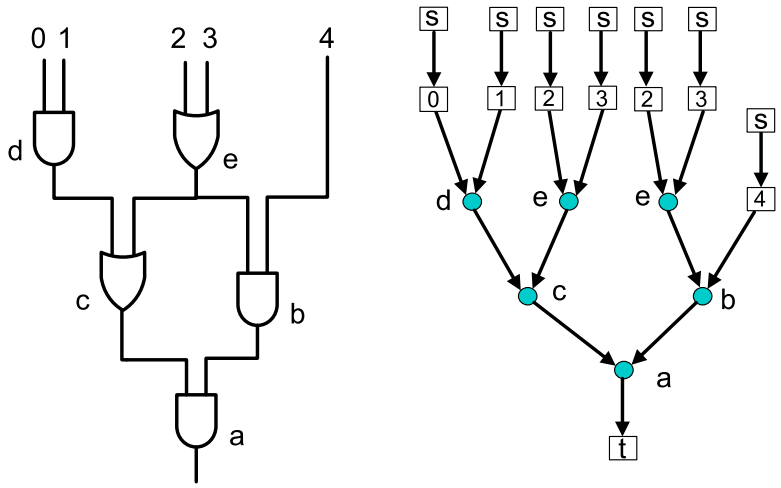


Figure 4.3: A simple circuit and the depth-first search tree

a simple circuit and the corresponding depth-first search tree. Readers can easily identify that the paths are already grouped depending on their suffixes in the depth-first search tree. We utilize this fact for the partitioning. In this example, we partition the set of all paths into the paths via $a \leftarrow c$ and the paths via $a \leftarrow b$ at the first round. At the second round, the paths via $a \leftarrow c$ are partitioned into the paths via $a \leftarrow c \leftarrow d$ and the paths via $a \leftarrow c \leftarrow e$. Also, the paths via $a \leftarrow b$ are partitioned into the paths via $a \leftarrow b \leftarrow e$ and the path $a \leftarrow b \leftarrow 4$ according to the depth-first search tree. In this type of partitioning, *the depth-first search tree is equivalent to the decision tree we desire to draw for the gambling*. In the decision tree, we assume that every node has at most two children (i.e., the vertices in the augmented timing graph have at most two incoming edges) without loss of generality. This is possible because we can modify the augmented timing graph by adding pseudo vertices.

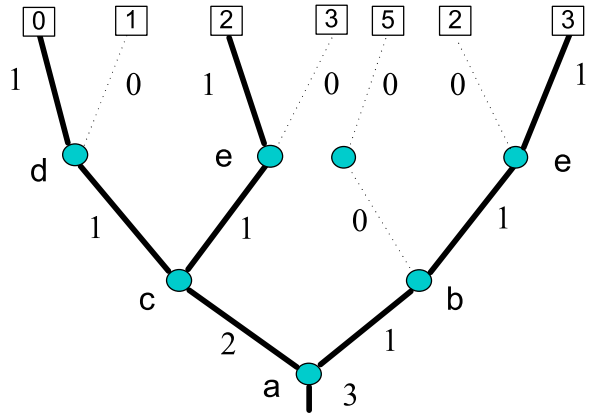


Figure 4.4: The decision tree with an example of bets

Figure 4.4 shows the decision tree with the bets of a gambler for the circuit in Figure 4.3. Each node in the decision tree corresponds to a game where the gambler bets on the two groups of the paths passing through each child. Simply, we can say that the gambler bets on the two children in each node. The initial capital is 3 and he divides it into 2 and 1 at the first game. If there exists a faulty path among the paths via $a \leftarrow c$, he receives 2 back and his capital also becomes 2. Since the capital and the payoff are the same, we can annotate one value for each node, and the value represents his capital in the case that there is a faulty path passing though the node. At the same time, the value represents his bet on the node in the game corresponding to the parent node. Each node in the decision tree can be uniquely identified by a subpath ending at the sink t. When we mention a subpath, it usually means a subpath ending at the sink t and readers can associate it with a node in the decision tree. Now we formally link this gamble with path selection.

4.4.2 Path Selection by Betting

Definition 6. *The bet on a subpath π is the number of paths to be selected among the paths passing through π.*

Definition 7. *The fault event of a subpath π is the event that a faulty path exists among the paths passing through π.*

We denote the bet and the fault event by B and F, respectively.

Definition 8. *The payoff μ of a subpath π is*

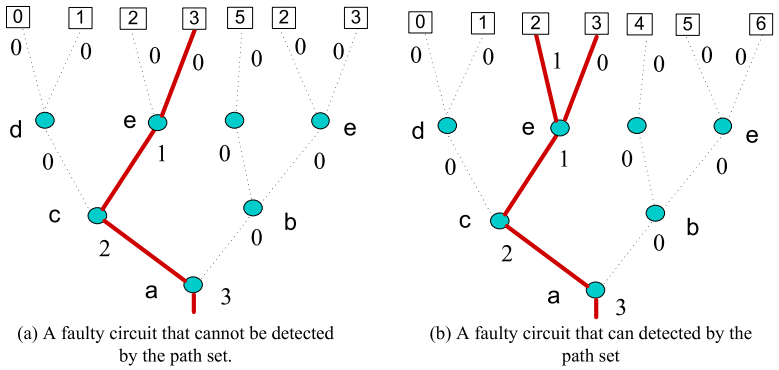
$$\mu(\pi) = \begin{cases} B(\pi) & \text{if } F_\pi \text{ occurs ;} \\ 0 & \text{otherwise.} \end{cases} \tag{4.7}$$

The definition of the bet immediately yields

$$B(t \leftarrow ... \leftarrow v) = \sum_{u \in N^-(v)} B(t \leftarrow ... \leftarrow v \leftarrow u). \tag{4.8}$$

It is important to note that the payoff and the bet for a subpath can be different as shown in the definition above, but in the decision tree, the fault event corresponding to each node is assumed to have occurred (this is what the decision tree represents) and the annotated value represents the bet and the payoff at the same time. Also note that the definition of the bets implies that the gambler selects the paths $a \leftarrow c \leftarrow d \leftarrow 0$, $a \leftarrow c \leftarrow e \leftarrow 2$ and $a \leftarrow b \leftarrow 3$.

Given a manufactured chip (an ensemble), we can depict the depth first search tree with the payoff of each node as shown in 4.5(a). Only along the



(a) A faulty circuit that cannot be detected by the path set.

(b) A faulty circuit that can detected by the path set

Figure 4.5: The payoffs for two produced chips (ensembles)

faulty path, the bets equal the payoffs. In this case, the gambler fails to reach a leaf of the tree which means that he fails to predict the faulty path. Note that the payoffs of all full paths (i.e., the payoffs of the leaf nodes) are zero. Figure 4.5(b) shows the DFS tree from another manufactured chip. In this chip, there are two faulty paths. For simplicity, we have explained the concept assuming there is at most one faulty path in a chip. However, multiple faulty paths do not affect our formulation, and for the concept, it can be viewed as several gamblers with each payoff play games in parallel. In the case of Figure 4.5(b), a gambler successfully reaches a leaf, predicting a faulty path. Note that there exist some paths with a non-zero payoff in this case. Using the payoff, we can re-write the objective function as

$$
\begin{aligned}
& P(\bigcup_{p\in\Pi} d(p) > T) \\
&= P(\bigcup_{p\in\Pi} \mu(p) > 0) = P(\sum_{p\in\Pi} \mu(p) > 0) \\
&= P(\sum_{p\in\Omega} \mu(p) > 0)
\end{aligned}
\tag{4.9}
$$

The last equality follows from the fact that $\mu(p) = 0$ for all $p \notin \Pi$. Let Ψ denote all subpaths ending at t. Using the new objective function, we can re-write the problem (4.6) as

$$
\begin{aligned}
& \max_{B(\cdot)} \; P(\sum_{p\in\Omega} \mu(p) > 0) \\
& \text{s.t.} \quad B(t) = k, \\
& \text{Equation (4.8) for all } \pi \in \Psi, \\
& B(\pi) \leq |\Omega_\pi| \text{ for all } \pi \in \Psi, \\
& B(\pi) \in \{0, 1, 2, 3, ...\} \text{ for all } \pi \in \Psi.
\end{aligned}
\tag{4.10}
$$

In the view of the gambler, a good anticipating strategy is required to solve this problem optimally. In other words, the gambler needs to account for future games before betting.

4.4.3 Betting Strategies

Considering the computational complexity of anticipating strategies, we will look for a casual strategy to solve this problem, i.e. the betting strategy of the gambler will depend on the results of the past games only. Due the use of a non-anticipating strategy, the gambler will have a local view which

is illustrated in Figure 4.6. Currently, his capital is $B(\pi)$ which equals $\mu(\pi)$ since it is a part of the decision tree. He will place bets on each side π_l and π_r depending on the probability of winning. He will not prefer one side against another due to the future games. Thus, if the probability of winning on each side is the same, he will just place the same bet on both sides. In this situation, it is reasonable to set his objective to maximize $\mu(\pi_l)+\mu(\pi_r)$, which is a random variable and its distribution changes according to $B(\pi_l)$ and $B(\pi_r)$. Figure 4.6

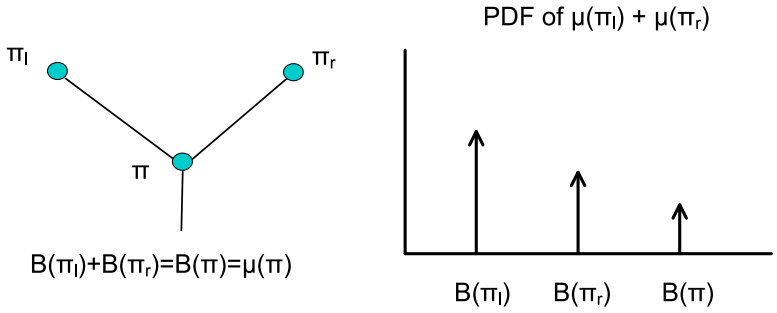


Figure 4.6: The local view of the gambler

shows a possible distribution of $\mu(\pi_l) + \mu(\pi_r)$. The probability values of the impulses are fixed, but under the constraint $B(\pi) = B(\pi_l)+B(\pi_r)$, the gambler can adjust the positions of the first two impulses. One adventurous gambler may prefer to place the impulse of $B(\pi_r)$ at $B(\pi)$, making the first impulse placed at 0. One conservative gambler may prefer to place the two impulses at $B(\pi)/2$. To make a choice among possible distributions, we need a measure to evaluate the distributions. One common measure for this is the expectation

$E[\cdot]$. Then, our strategy can be written as

$$\begin{array}{ll} \max\limits_{B(\pi_l),B(\pi_r)} & E[\mu(\pi_l)+\mu(\pi_r)] \\ \text{s.t.} & B(\pi_l)+B(\pi_r)=B(\pi). \end{array} \tag{4.11}$$

In our betting strategy, we will ignore the constraint on the numbers of paths in Ω_{π_l} and Ω_{π_r}, and this will be handled in a different way later. A simple calculus results in that if the probability of $B(\pi_l)$ is greater than that of $B(\pi_r)$, $B(\pi_l) = B(\pi)$ and $B(\pi_r) = 0$. Otherwise, $B(\pi_l) = 0$ and $B(\pi_r) = B(\pi)$. Clearly, this is too aggressive and will lead to bankruptcy almost certainly after a few games. To deal with this issue, we need to consider that the gambler will play a sequence of games, and a gambler should guess correctly in all the games in order to predict a faulty path. Thus, a more conservative strategy is required. This issue also arises in real-life gambles and investments. Compared to real-life ones, our gamble is a bit restrictive in the sense that the payoffs from π_l and π_r are not put together to play the next game. However, they share a more important common aspect. In real-life gambles, the capital of a gambler after n rounds of games is the product of factors which also hold in our case when we define an indicator function of the fault event. If one factor is zero, the final result becomes zero as well, so we need to be conservative. Since this is the basis of the portfolio theory derived from the information theory, we adopt a strategy called the *log-optimal portfolio* from the portfolio theory which is to use the log expectation [19, 33]. In a series of real-life games represented by independent and identically distributed (i.i.d.) random variables, the log-optimal portfolio provides asymptotically optimality among casual strategies. In a non-i.i.d. case, the *conditionally log-optimal portfolio*

is known to provide a comparable solution to any other causal portfolios [19]. Before writing our strategy, we will relax the integer constraints on each bet, and we define the *betting fraction* of a subpath as

$$\alpha(t \leftarrow ... \leftarrow v \leftarrow u) = \frac{B(t \leftarrow ... \leftarrow v \leftarrow u)}{B(t \leftarrow ... \leftarrow v)}. \tag{4.12}$$

Then, our strategy can be written as

$$\begin{array}{ll} \max\limits_{\alpha(\pi_l),\alpha(\pi_r)} & E[log\{\mu(\pi_l) + \mu(\pi_r)\}] \\ \text{s.t.} & \alpha(\pi_l) + \alpha(\pi_r) = 1. \end{array} \tag{4.13}$$

This is a constrained optimization problem and the Kuhn-Tucker conditions give a necessary condition to the optimal betting fraction α^*. Due to the concavity of *log*, the condition also becomes sufficient. Our problem is nearer to the non-iid case, and the condition is a function of conditional probabilities of the fault events. To compute these probabilities easily, we assume that even if there are faulty paths in both π_l and π_r, this knowledge is not given to the gambler in the future games. Let us define the *fault probability*, which is the probability that the fault event occurs and is denoted by f. Then we can write

$$f(t \leftarrow \ldots \leftarrow v) \equiv P(F_{t \leftarrow ... \leftarrow v}). \tag{4.14}$$

We also define the *joint fault probability* by

$$f_j(t \leftarrow \ldots \leftarrow v) \equiv P\left(\bigcap_{u \in N^-(v)} F_{t \leftarrow ... \leftarrow v \leftarrow u}\right). \tag{4.15}$$

Then, the optimal betting fractions can be written as

$$\begin{aligned} \alpha^*(\pi_l) &= \frac{f(\pi_l) - f_j(\pi)}{f(\pi_l) + f(\pi_r) - 2f_j(\pi)}, \\ \alpha^*(\pi_r) &= 1 - \alpha^*(\pi_l). \end{aligned} \tag{4.16}$$

The following two propositions shows that the fault probability and the joint fault probability can be computed easily using the maximum operation provided in a SSTA tool.

Proposition 1. *Let π be a path $t \leftarrow \ldots \leftarrow v$. Then,*

$$f(\pi) = P(d(t \leftarrow \ldots \leftarrow v) + at(v) > T). \tag{4.17}$$

Proof.

$$\begin{aligned} f(t \leftarrow \ldots \leftarrow v) &= P(\bigcup_{p \in \Omega_{t \leftarrow \ldots \leftarrow v}} d(p) > T) \\ &= P(\max_{p \in \Omega_{t \leftarrow \ldots \leftarrow v}} \{d(p)\} > T) \\ &= P(d(t \leftarrow \ldots \leftarrow v) + at(v) > T). \end{aligned} \tag{4.18}$$

The last equality follows from the distributivity of the maximum operation over the addition operation, $\max(A + B, A + C) = A + \max(B, C)$, and the fact that all paths in $\Omega_{t \leftarrow \ldots \leftarrow v}$ share the same suffix $t \leftarrow \ldots \leftarrow v$. □

Proposition 2. *Let π be a path $t \leftarrow \ldots \leftarrow v$. Then,*

$$\begin{aligned} &f_j(\pi) = \\ &\quad P(d(t \leftarrow \ldots \leftarrow v) + \min_{u \in N^-(v)} \{d(v \leftarrow u) + at(u)\} > T). \end{aligned} \tag{4.19}$$

Proof.

$$
\begin{aligned}
& f_j(t \leftarrow \ldots \leftarrow v) \\
& \quad = P\left(\bigcap_{u \in N^-(v)} F(t \leftarrow \ldots \leftarrow v \leftarrow u)\right) \\
& \quad = P\left(\bigcap_{u \in N^-(v)} d(t \leftarrow \ldots \leftarrow v \leftarrow u) + at(u) > T\right) \\
& \quad = P(\min_{u \in N^-(v)} \{d(t \leftarrow \ldots \leftarrow v \leftarrow u) + at(u)\} > T) \\
& \quad = P(d(t \leftarrow \ldots \leftarrow v) + \min_{u \in N^-(v)} \{d(v \leftarrow u) + at(u)\} > T).
\end{aligned}
\tag{4.20}
$$

□

Note that $min(A, B)$ can be easily computed by $-max(-A, -B)$ in statistical static timing analysis. Also note that the left hade side quantities of the inequalities in (4.17) and (4.19) are represented in the form of (1.1), and (4.17) and (4.19) are reduced to (1.2).

4.4.4 Summary

Algorithm 5 summarizes the overall procedure. Initially, *TopDownSelection* is called with $\pi = t$, $v = t$ and $m = k$. Then, it will return a set Π of k best paths. The algorithm performs the recursive depth-first search (DFS) from the sink t. For each branch in the depth-first search tree, it computes the optimal betting fraction by (4.16), (4.19) and (4.17) (line 7), and each branch is traversed only if a non-zero bet is placed on the branch (line 8). This algorithm only traverses a small part of the DFS tree because the capital is exhausted quickly.

Algorithm 5 TopDownSelection

Input: π, v, m
Output: Π

```
1: Π ← ∅
2: if v is a start point then
3:    Add π to Π
4: else
5:    for u ∈ N⁻(v) do
6:       Extend π to u and let π′ denote the extended path
7:       B(π′) ← α*(π′) × m
8:       Discretize B(π′) s. t. the sum of B(π′)s through this iteration equals
         m
9:       if B(π′) > 0 then
10:         Add TopDownSelection(π′, u, B(π′) ) to Π
11:      end if
12:   end for
13: end if
14: return Π
```

4.5 Guaranteeing k Paths and Handling Untestable Paths

Since the available number of paths is not taken into account, some paths get a bet more than 1, and Algorithm 5 may produce less than k paths. If it selects less than k paths, we may increase k to select more paths. However, in order to guarantee k paths, time consuming iteration is required until k paths are selected. For this problem, we observe the fact that paths are selected in a particular order as k increases. If the discretization is not performed, the bet for each path π can be written as

$$B(\pi) = B(t) \prod_{i=0}^{n} \alpha^*(\pi_i) \tag{4.21}$$

where n is the length of π and π_i is the subpath from the i-th segment of π to t. We can rank paths by the amount of bets, and this ranking does not depend on the initial capital $B(t)$. However, according to our experiments, the discretization is helpful to increase the quality of results, and we develop an algorithm that produces the same path set as in Algorithm 5. A possible implementation of the dicretization in Algorithm 5 is as follows:

$$\begin{aligned} B(\pi_l) &= \lfloor B(\pi)\alpha^*(\pi_l) + 0.5 \rfloor, \\ B(\pi_r) &= \lceil B(\pi)\alpha^*(\pi_r) - 0.5 \rceil. \end{aligned} \tag{4.22}$$

We compute $B(\pi_l)$ and $B(\pi_r)$ from $B(\pi)$ in Algorithm 5. Conversely, given $B(\pi_l)$, we can obtain the range of $B(\pi)$ by

$$\frac{B(\pi_l) - 0.5}{\alpha^*(\pi_l)} \leq B(\pi) < \frac{B(\pi_l) + 0.5}{\alpha^*(\pi_l)}. \tag{4.23}$$

Similarly, given $B(\pi_r)$, we can get the range of $B(\pi)$ by

$$\frac{B(\pi_r) - 0.5}{\alpha^*(\pi_r)} < B(\pi) \leq \frac{B(\pi_r) + 0.5}{\alpha^*(\pi_r)}. \tag{4.24}$$

In the case that $B(\pi)$ increases from zero, the subpaths π_l and π_r start receiving $B(\pi_l)$ and $B(\pi_r)$ when

$$\begin{aligned} B(\pi) &= \left\lceil \frac{B(\pi_l) - 0.5}{\alpha^*(\pi_l)} \right\rceil \text{ and} \\ B(\pi) &= \left\lfloor \frac{B(\pi_r) - 0.5}{\alpha^*(\pi_r)} + 1 \right\rfloor, \end{aligned} \tag{4.25}$$

respectively. This allows us to run the procedure in Algorithm 5 backward without the initial bet $B(t)$. Suppose that for π_l (π_r), we have a set of pairs of a path and an integer which represent that the path is selected when the

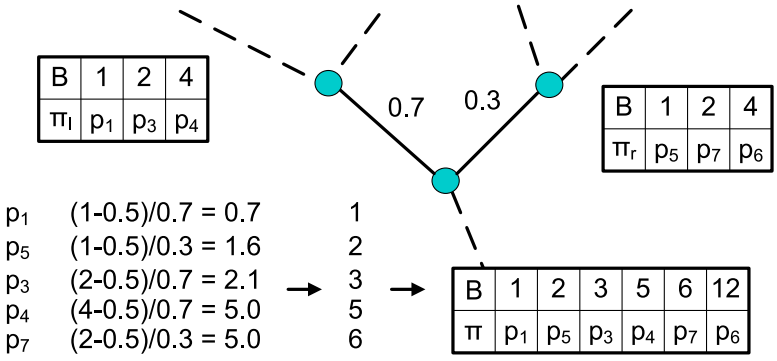


Figure 4.7: Merge process

bet on π_l (π_r) reaches the integer. Then, we can merge the two sets into a set, converting the integers for the bet on π. Figure 4.7 illustrates the merge process. Suppose that if the bet for π_l is increasing from 1 to 4, then p_1, p_3, and p_4 are selected in the order named. No additional path is selected for bet 3. This is tabulated in Figure 4.7, where the table for π_r is also shown. For each p_1, p_3, and p_4, $B(\pi)$ can be obtained from $\alpha^*(\pi_l)$ and the corresponding $B(\pi_l)$ by (4.25). Similarly, $B(\pi)$ can be obtained for each p_5, p_7, and p_6. The two path sets are merged in the increasing order of $B(\pi)$ and become a path set for π. Algorithm 6 summarizes the new procedure for guaranteeing k paths. This is a divide and conquer (DnC) algorithm (e.g., merge sort). The paths in the circuit are recursively divided into two parts based on their suffixes, and the two subprograms are solved independently (line 11 and 14) and the solutions are merged together (line 16). This suffix-based divide and conquer approach is also useful to exclude untestable paths from consideration using

the fact that if a subpath is not testable, all the paths passing though the subpath are not testable. In the top-down phase of our DnC algorithm, we check if the subpath π is testable using a specialized ATPG tool such as [26] or incremental SAT [35] (line 1). If the subpath π is not testable, all paths passing through π are untestable, and we can prune the branch. The merge process is detailed in Algorithm 7.

Algorithm 6 BottomUpSelection

Input: $\pi, k; \epsilon$

1: **if** π is not testable **then**
2: **return**
3: **end if**
4: Let v be the last vertex of π
5: **if** v is a start point **then**
6: $\Pi \leftarrow \Pi \cup (1, \pi)$
7: **else**
8: Let π_l and π_r be extended paths toward the two vertices in $N^-(v)$, respectively
9: $\Pi_l, \Pi_r \leftarrow \emptyset$
10: **if** $\alpha(\pi_l) < \epsilon$ **then**
11: $\Pi_l \leftarrow$ BottomUpSelection (π_l, k)
12: **end if**
13: **if** $\alpha(\pi_r) < \epsilon$ **then**
14: $\Pi_r \leftarrow$ BottomUpSelection (π_r, k)
15: **end if**
16: $\Pi \leftarrow$ MergePathSet(π_l, Π_l, π_r, Π_r, k)
17: **end if**
18: **return** Π

Algorithm 7 MergePathSet

Input: $\pi_l, \Pi_l, \pi_r, \Pi_r, k$
Output: Π

$I \leftarrow 0, J \leftarrow 0$
$\Pi \leftarrow \emptyset$
for $m = 0$ to $\min(k, |\Pi_l| + |\Pi_r|)$ **do**
  Let (B_l, p_l) be I-th element of Π_l
  Let (B_r, p_r) be J-th element of Π_r
  $\hat{B}_l \leftarrow \lceil (B_l - 0.5)/\alpha^*(\pi_l) \rceil$
  $\hat{B}_r \leftarrow \lfloor (B_r - 0.5)/\alpha^*(\pi_r) + 1 \rfloor$
  if $\hat{B}_l \leq \hat{B}_r$ **then**
    Add $(\hat{B}_l, p_l)$ to Π
    $I \leftarrow I + 1$
  else
    Add $(\hat{B}_r, p_r)$ to Π
    $J \leftarrow J + 1$
  end if
end for
return Π

4.6 Experimental Results

We implemented the proposed method in C++, and used ISCAS85 circuits and or1200 [1] as benchmarks. The or1200 includes a 32-bit, 5-stage Wallace multiplier. To check the testability, *minisat* [21] was incorporated in our implementation. All experiments were performed on a 3.0GHz 2 Xeon X5570 Linux machine. All benchmarks are technology-mapped to TSMC 180nm library. The nominal delay annotated to each edge in the timing graph is the sum of the gate and wire delays obtained from the Standard Delay Format (SDF) file.

Our statistical static timing analysis (SSTA) tool uses the algorithm proposed in [71]. To model spatial correlation, we use a quad tree [4] with 3 levels which results in 21 global sources of variation. The global sources of variation associated with the first, the second and the third level can change the delay of each edge up to $\pm 4\%$, $\pm 5\%$ and $\pm 6\%$ of its nominal value in the 3σ case, respectively. The delay of each edge also has independent variation, and the 3σ point is 5% of the nominal delay.

In our experiments, we assume that non-robust tests can determine the delay of the path being targeted irrespective of other path delays as in robust tests. We compare our method to the two branch and bound methods (BnB-JPM and BnB-SPM) proposed in [89] and top k-longest path selection in deterministic timing analysis (Deterministic). Given a number of paths to be selected k, each method constructs a path set Π of size k, and the PCM value $q(\Pi)$ of the path set is calculated by a Monte Carlo simulation with

Table 4.1: Pre-ATPG PCM and runtimes for ISCAS85 circuits

	Deterministic		BnB-JPM [89]		BnB-SPM [89]		Proposed	
	q(Π)	CPU(s)	q(Π)	CPU(s)	q(Π)	CPU(s)	q(Π)	CPU(s)
c432	0.990	0.000	0.946	0.006	0.991	0.002	0.991	0.004
c499	0.674	0.004	0.831	0.023	0.872	0.006	0.906	0.011
c880	0.983	0.001	0.969	0.004	0.983	0.004	0.999	0.004
c1355	0.554	0.005	0.799	0.046	0.811	0.007	0.876	0.011
c1908	1.000	0.001	0.979	0.012	1.000	0.005	1.000	0.006
c2670	0.993	0.005	0.968	0.014	0.998	0.010	0.999	0.009
c3540	0.950	0.002	0.969	0.015	0.950	0.011	1.000	0.010
c5315	0.981	0.006	0.980	0.021	0.981	0.019	0.998	0.020
c6288	0.995	0.008	0.995	0.166	0.995	0.045	0.995	0.816
c7552	1.000	0.006	1.000	0.022	1.000	0.025	1.000	0.021

100,000 samples.

We first assume that all paths in the circuit are testable as in most path selection studies [72, 89, 16] and for each ISCAS85 circuit, each method selects 5 paths except c499, c1355 and c2670. Since these circuits have many-near critical paths, 30 paths are selected instead. Table 4.1 shows the PCM values (q(Π)) and the runtime values (CPU). The basic idea of top k longest path selection is to use the fact that long paths are more likely to fail to meet the timing than shorter ones. BnB-SPM explicitly computes the fault probabilities using the statistical timing model and provides better path sets than the top k longest path selection. Our proposed method goes further and utilizes statistical correlation of the delay variations intelligently and produces path sets of better quality than BnB-SPM at a small runtime overhead. The benefit of smart path selection is more significant in highly optimized industrial

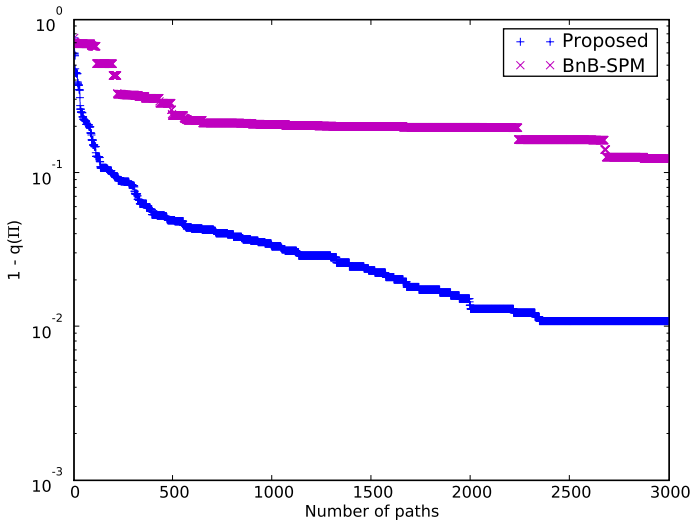


Figure 4.8: Pre-ATPG PCM

circuits since they have numerous near-critical paths which offer many possible choices. Figure 4.8 shows 1-q(Π) as k increases for or1200. The value 1-q(Π) is also called the *missing probability* in [72]. If a smart path selection algorithm is employed, good paths are selected first, and it is natural to exhibit diminishing returns as k increases. In other words, a small improvement in a high PCM value requires testing a larger number of paths than that in a low PCM value. In order to account for this behavior, we plot this type of graphs in a logarithmic scale. In or1200, the proposed method shows a significant improvement of the coverage over BnB-SPM at the same number of paths, or a significant reduction of the required number of paths for a given PCM value. In the conventional flow, the paths selected in this way are fed into an automatic test pattern generation (ATPG) tool which generates test patterns sensitizing

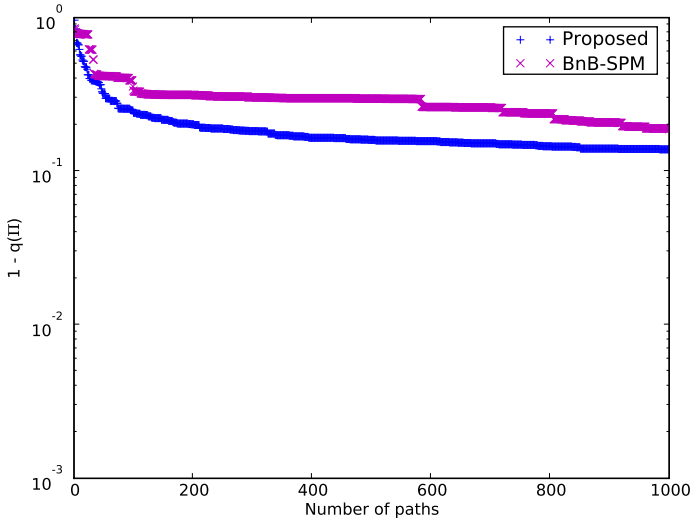


Figure 4.9: Post-ATPG PCM

the target paths. During this process, some paths are determined as untestable and the quality loss (i.e., coverage loss) occurs. The quality loss can be more severe in the path set produced by the proposed method. This is because BnB-SPM covers the process parameter space redundantly, while our method tries to cover as large area as possible with a small number of paths and in order to achieve this goal, it avoids redundancy. However, the redundancy can actually mitigate the testability problem. Balancing this redundancy and efficiency in a smart way is proposed in [74]. However, it does not still guarantee that the quality loss does not occur, and we circumvent this testability issue completely by integrating the minisat solver. Our method as well as the methods in [89] can consider only testable paths as candidate paths in this way. Figure 4.9 shows post-ATPG PCM values using this approach. The proposed method

Table 4.2: Post-ATPG PCM and runtimes for ISCAS85 circuits

	BnB-SPM [89]		Proposed	
	q(Π)	CPU(s)	q(Π)	CPU(s)
c499	0.868	0.021	0.888	0.071
c880	0.983	0.004	0.999	0.005
c1355	0.851	0.027	0.878	0.085
c2670	0.998	0.027	0.999	0.019
c5315	0.996	0.030	0.998	0.045
c432	0.004	0.006	0.004	0.009
c1908	0.000	0.055	0.000	0.017
c3540	0.012	0.074	0.012	0.041
c6288	0.034	57.280	0.034	118.108
c7552	0.005	0.033	0.005	0.028

still provides better quality than BnB-SPM, but compared to the pre-ATPG PCM values (i.e., the PCM values when assuming all paths are testable), the improvement is marginal. The PCM values seem to approach a certain limit. Actually, in many designs, the post-ATPG PCM value does not become one even if all testable paths in the design are tested.

Table 4.2 shows the post-ATPG results for ISCAS85 circuits. For the top 5 circuits, our proposed method provides better post-ATPG coverage than BnB-SPM. BnB-SPM also shows adequate PCM values for 5 paths. However, for the other 5 circuits, both methods result in path sets of unreasonable quality. There are two possible explanations for this result and the limit in or1200. First, the circuits may have false paths. In this case, the selected paths may be actually good, but the test metric q(Π) fails to measure the quality of the path sets correctly. This issue arises because q(Π) is defined

Table 4.3: Post-ATPG PCM and runtimes for ISCAS85 circuits using test margin

		BnB-SPM [89]			Proposed		
	TestMargin	FalseNegative	q(Π)	CPU(s)	FalseNegative	q(Π)	CPU(s)
c499	0%	0%	0.868	0.021	0%	0.888	0.071
c880	0%	0%	0.983	0.004	0%	0.999	0.005
c1355	0%	0%	0.851	0.027	0%	0.878	0.085
c2670	0%	0%	0.998	0.027	0%	0.999	0.019
c5315	0%	0%	0.996	0.030	0%	0.998	0.045
c432	3%	2.6%	0.817	0.015	2.6%	0.817	0.032
c1908	13%	1.6%	0.992	0.125	1.6%	0.992	0.260
c3540	3%	1.9%	0.998	0.075	1.9%	0.998	0.193
c6288	2%	0.2%	0.895	81.382	0.2%	0.895	210.865
c7552	3%	1.7%	0.945	0.038	1.7%	0.945	0.038

without the consideration about false paths. To deal with this issue, we need to eliminate false paths in timing analysis by specifying them manually or by an automated tool. This process can immediately improve q(Π) for the same path sets. Secondly, there may exist some true paths that are not testable. Such paths are common in any circuits, and the constraints that arise from scan types (e.g., launch on capture, launch on shift, etc) increase the number of such paths, which are not considered in this experiment. If such paths cause poor quality, we can test the target paths faster-than-at-speed (i.e., we can use a shorter clock period for testing than the operating clock period). The difference between the operating clock period and the test clock period is called the *test margin*. A systematic approach to determine the optimal test margin is proposed in [78]. For the rest of the experiments, we will assume that the test quality is degraded after ATPG for the second reason.

Table 4.3 shows the post-ATPG PCM values using the test margin. The test margins are expressed by the percentage to the operating clock period. For the top 5 circuits, we do not use the test margin because the PCM values are satisfactory considering only 5 paths are selected. In this case, there is no false negative (i.e., yield loss) under our model because chips are determined as bad ones only when actual faulty paths are found. The amount of false negative chips is expressed by the percentage to the number of produced chips. For the bottom 5 circuits, we use non-zero test margins to obtain decent PCM values, and both methods successfully achieve it. However, the proposed method does not provide better quality of results than BnB-SPM. Actually they select the

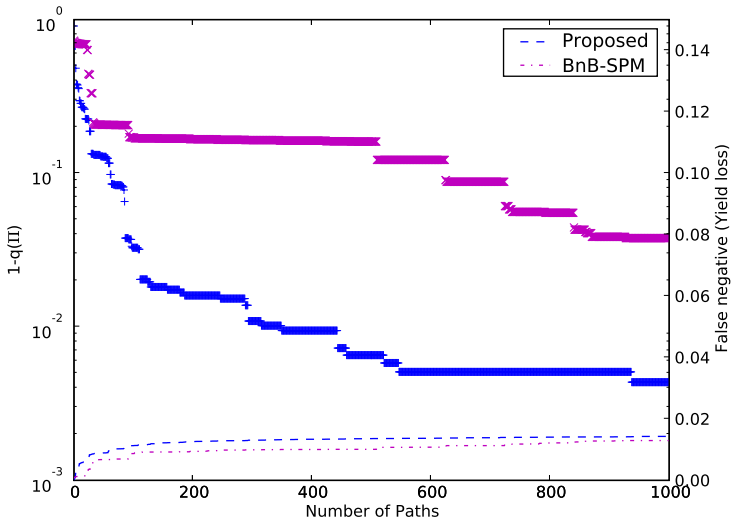


Figure 4.10: Post-ATPG PCM when both methods use 0.7% test margin

same paths, and this is because there are not many near-critical and testable paths in these circuits. In this scenario, the selection simply does not matter. Fortunately, this scenario is uncommon in highly optimized large industrial circuits. Figure 4.10 shows the PCM values and the ratio of false negative chips when 0.7% test margin is used. The improvement of the proposed method shown in Figure 4.8 has disappeared in Figure 4.9 and now appears back in Figure 4.10. The high quality is obtained at the cost of false negative, but the proposed method does not incur additional yield loss in comparison with BnB-SPM. Readers can identify this by checking that the amount of false negative is similar at the same PCM value. This becomes clearer when we use a more aggressive 1% test margin for the path set from BnB-SPM to make up for the insufficient coverage. Figure 4.11 shows this scenario. The

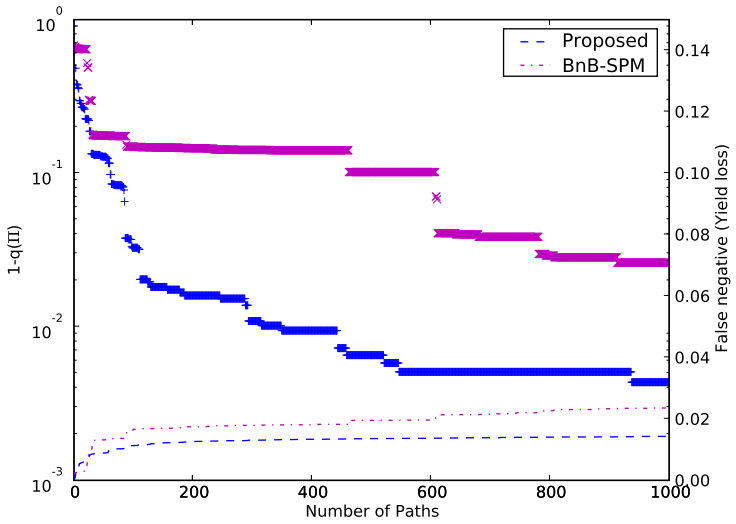


Figure 4.11: Post-ATPG PCM when proposed methods and BnB-SPM use 0.7% and 1% test margin, respectively

coverage is improved slightly but in this case, our proposed method provides a significantly better PCM value and a better yield using the same number of paths, or the same PCM value and better yield using a significantly smaller number of paths.

4.7 Conclusions

In order to deal with the increasing variability of circuit delays and the growing impact of small delay defects, we have presented a new class of statistical path selection algorithm to generate a compact, high quality path set. Our algorithm outperforms existing methods utilizing statistical correlation of the delay variations intelligently. Most studies for path selection ignore

the testability issue with the hope that the improvements in pre-ATPG quality would be maintained after ATPG. Our experimental results have revealed that in some designs, this is the case, but there exists other designs where the improvement becomes marginal after ATPG. We have also found that if this is not the problem of the test metric due to false paths, faster-than-at-speed test can recover the original improvement. If it is the test metric issue, false paths should be eliminated from the timing analysis. This process is usually done before path selection for correct timing analysis but is often incomplete and cumbersome. Thus, we may need an alternative test quality metric which may be our future work.

Chapter 5

Testability Driven Statistical Path Selection

5.1 Introduction

The selection of paths to be targeted in delay testing has been an issue over the past decades, and various path selection methods have been proposed. Some of them target localized delay faults due to timing defects [63], and others target accumulated delay faults due to process variations [47]. To deal with ever-increasing process variation, a lot of effort has been devoted to developing a model for the variation and efficient algorithms for analyzing timing behavior on top of the model [71]. Since the modeling and analysis techniques are becoming mature, path selection algorithms that leverage the statistical timing framework are gaining more attention [89, 16]. These algorithms take advantage of the global and spatial correlations between delays and generate a higher quality path set with a smaller number of paths. Previous approaches are not aware of the correlations or use them in an ad-hoc manner, while the recent algorithms [89, 16, 71, 77] can quantify them using the statistical timing model.

Statistical path selection algorithms can be categorized as *covering-based algorithms* [89, 16] and *criticality-based algorithms* [71, 77]. Covering-

based algorithms solve the well-known *maximum coverage* problem in the process parameter space. In the process parameter space, the test of a path delay covers a region, and the objective of the algorithms is to cover as large area as possible with a limited number of paths. Criticality-based algorithms rank paths by the criticality, which is the probability that a path become the critical (longest) path. Covering-based algorithms take a test clock period as input and try to find a specifically optimized solution for the given test clock period, while criticality-based algorithms does not require the test clock period and produces a path set that is good irrespective of the test clock period. They are thus suitable for speed-binning where a test set is run at various frequencies.

Once the paths to be tested are determined by one of these algorithms, they are fed into an ATPG (Automatic Test Pattern Generation) tool, which generates the test patterns sensitizing the paths. During the ATPG process, some paths turn out to be untestable which results in test coverage loss. Since many physical paths can be untestable [25, 27], this coverage loss is a severe problem. To make up for the loss, more paths may be selected and targeted in ATPG, and this process can be continued until a desired test coverage is achieved. However, the number of the iterations can be very large and the iterations that involve running separate tools take a lot of time. In particular, if the path selection algorithm used does not support incremental selection, the path selection time gets longer as the process continues.

In the covering-based path selection, several approaches are proposed to deal with that issue. In [74], the authors propose a method that covers the

parameter space m times, where m is provided by a user. In this method, the test coverage loss does not happen unless the m paths that cover a region are untestable at the same time. In [73], the authors use the paths which are found in the path selection process but are discarded because they cover almost the same regions as already selected paths. The proposed approach creates a hierarchy of paths, and the children of a path are used as alternatives to the path since the area covered by a child is contained in that of the path. Both methods are efficient and can leverage conventional tools. However, they require a decent ratio of testable paths to the selected paths and do not guarantee the selection of a required number of testable paths. The authors in [16] show that their covering-based algorithm, combined with a SAT solver, can ensure k testable path selection, but the test quality metric used in [16] is affected by false paths and actual post-ATPG results are not presented.

The testability issue in criticality-based path selection cannot be dealt with merely by integrating a SAT solver and requires a new selection algorithm, which is presented in this chapter. The major contributions of this chapter can be summarized as follows.

- Most statistical test quality metrics in the literature as well as that in [16] ignore false paths, which cripple the metrics. We define the (critical) testable path coverage metric in the process parameter space, and the test quality is measured independent of the existence of false paths.

- Our criticality-based algorithm guarantees the selection of k testable

paths, and unlike [16, 74, 73], our technique can also be used for speed-binning.

- Our algorithm is based on the branch and bound framework proposed in [89], and we show that the efficient pruning of the framework is still applicable in the criticality metric. Also, the algorithm in [89] requires $O(kN)$ time, whereas our algorithm can run in $O(N)$ time, where N is the number of paths after pruning and k is the number of paths to be selected.

- Instead of using incremental SAT for the testability check as in [35], we propose an alternative method, reducing the number of the SAT calls substantially. This new integration method can be applied to other applications that require solving the ISAT problem defined in [35].

5.2 Background

In this section, we first define basic terms used throughout this chapter. The arrival time, the required arrival time and the delay are denoted by AT, RAT and d, respectively. The timing slack $S(p)$ of a (sub) path p from vertex v to w is defined as

$$S(p) = RAT(w) - AT(v) - d(p). \tag{5.1}$$

The slack of a path set U is the minimum of the path slacks in U and is denoted by S_U. The set of all paths in the given circuit is denoted by Ω. The *chip*

slack is the minimum of the slack of all paths in the circuit, and is denoted by S_Ω. If a path cannot be *sensitized* by any vector pairs, the path is a *false path*. The viability [52] and floating-mode condition [14] are commonly used sensitization criteria. Delay faults in false paths do not affect the functionality of the circuit. *Robustly testable* paths can be tested irrespective of the other path delays, while *non-robustly testable* paths can be tested only if some paths providing the initial value are not slow. The *functionally un-sensitizable* paths are a sufficient condition for false paths, and the *statistically sensitizable* paths equivalent to non-robustly testable paths are a sufficient condition for true paths. The functionally sensitizable paths can affect circuit timing only if some other paths become slow and are sensitized at the same time. Thus they are not testable by a pair of vectors in most cases. The formal definitions of these conditions are shown in [14]. The *true chip slack* is the slack of the set of true paths, and it is less than or equal to the slack of the set of all testable paths and greater than or equal to the chip slack.

5.2.1 Deterministic vs. Statistical Path Selection

The selection of paths to be targeted in at-speed test has been done though deterministic STA. This deterministic path selection has several drawbacks compared to statistical path selection. First, path selection is done in a particular corner, usually the worst case corner. The critical or near-critical paths in other regions of the process space can be very different from those in the worst case corner. Second, statistical path selection accounts for the

sensitivities of path delays to various uncertain parameters. For example, suppose that we are to select one out of path A and B. Also suppose that path A and path B have the same nominal delay but path A has a larger sensitivity to the thickness of metal 1 than that of path B. It is clear that we should select path A for a better test quality, but deterministic path selection does not distinguish path A and B. Also, statistical path selection works on an abstracted model and it can be generally used for various sources of variation (e.g., metal 1, metal 2, Vdd, aging effects, etc, or any combinations of those). The sensitivities considered in statistical path selection are determined by the statistical timing model used, and we do not have to develop a particular path selection algorithm for each source of variation. Third, deterministic path selection does not leverage statistical correlations of path delay variations. For example, deterministic path selection may select paths sensitive to metal 1 only, whereas statistical method can avoid that because we may be able to obtain enough information about metal 1 from a few metal-1-sensitive-paths. Similar scenarios occur in many cases. A deterministic method may select paths in a particular region of a die, a particular module, or a particular part of a circuit (e.g., a chain of reconvergent paths). On the other hand, statistical path selection ensures diversity to increase the test quality.

5.3 Problem Formulation

Our objective is to construct a set of potentially longest testable paths in a design. In practice, the longest path means the least slack path and in this

chapter we use them interchangeably. In the deterministic timing model, the iterative process of the top k longest path selection and ATPG can enumerate testable paths in the order of their lengths, but in the statistical timing model, the selection of the top k paths with highest criticality and the following ATPG will fail to list testable paths in the order desired.

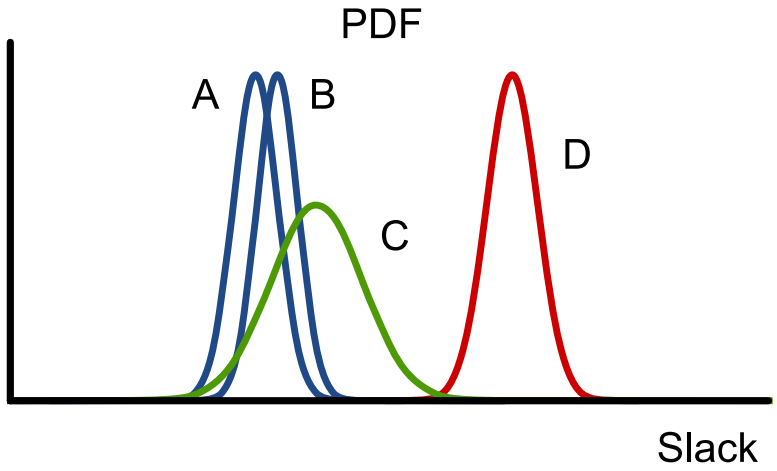


Figure 5.1: Slack distributions of 4 paths

Let us consider a circuit with 4 paths, and suppose that Figure 5.1 shows the PDFs (probability density functions) of the path slacks. Also suppose that A and B are perfectly correlated and other slacks are independent of each other. Out of the 4 paths, we will select 2 paths for at-speed test. In this example, we will consider two different scenarios on the testability of each path. First, we assume that all the paths are testable. If we select two longest path using deterministic STA, A and B will be selected. However, since A and B are perfectly correlated, and their slacks change together, path B will not be the critical path in any realizations (i.e., any produced chips). Thus it is

redundant to test path B and some test resources are wasted. If we select the top two paths with the highest criticality using statistical STA, A and C will be selected. This is because SSTA knows the correlation and the criticality of B becomes zero. Thus we can more effectively test the circuit with the same test budget. Now, we consider the case where path A is not testable and the other paths are testable. The following ATPG process will determine that A is not testable, and one more path is queried to the statistical STA tool which will return path D since the criticality of B is zero. In the end, C and D are tested and the test quality is degraded. Readers can easily identify B and C as the best paths, but SSTA fails to select them.

5.3.1 Testable Path Coverage Metric

To get around this issue, we confine our solution space to testable paths from the path selection stage and develop a new quality metric generalizing the path criticality concept.

Definition 9. *A path p is critical in a path set U if $S(p) \leq S(s)$ for all $s \in U$.*

Definition 10. *The criticality of a path p in a path set U is the probability that a path p is critical in U.*

The criticality in a path set U is denoted by λ_U. Then, the conventional path criticality becomes λ_Ω. Let Ω_t denote the set of all (robustly or non-robustly) testable paths. For a given path set Π, *(critical) testable path coverage metric* (TCM) is defined as the probability that the path set Π con-

tains the longest testable path and is denoted by $q(\Pi)$. Then, we can write

$$q(\Pi) = P(\bigcup_{p \in \Pi} p \text{ is critical in } \Omega_t). \tag{5.2}$$

We will compare TCM with other test metrics in the existing work. The set of tested paths is denoted by Π and we call S_Π the *test slack*. The correlation coefficient of the chip slack and the test slack used in [46] can also be a good metric to evaluate a path set for at-speed testing. However, since a true or false path is defined based on delay values, the true chip slack cannot be simply computed by the statistical minimum of the slacks of a certain set of paths. The chip slack computed by SSTA is an upper bound of the true chip slack, and we can use it for the correlation coefficient, but the sub-optimality even under the timing model is unavoidable in practice. To compare TCM to the other test metrics, we first consider the probability that a DUT (device under test) is classified to a correct bin (i.e., operating frequency). Also, for the comparison, we assume that there is only one bin, the test margin (i.e., guard band) is zero and all paths are testable as in the previous studies where the metrics to be compared are defined. Then, we can write

$$\begin{aligned} &P(Correctly\_classified) = \\ &\qquad P(S_\Pi < 0 | S_\Omega < 0) P(S_\Omega < 0) + P(S_\Omega > 0) \end{aligned} \tag{5.3}$$

where $P(S_\Pi < 0 | S_\Omega < 0)$ is called PCM (*process space coverage metric*) in [74] (a.k.a., the detection probability in [16]) and it is a statistical counterpart to the conventional fault coverage. $P(Correctly\_classified)$ has a simple linear relation with the PCM. Given $S_\Omega < 0$, it is a sufficient condition for $S_\Pi < 0$

that Π contains the critical path. Thus we obtain

$$P(Correctly\_classified) \geq q(\Pi). \tag{5.4}$$

With TCM, our objective becomes to maximize $q(\Pi)$ subject to $|\Pi| = k$ where k is the number of paths to be selected. In this problem, unless the size of the path set should be constrained, it leads to a trivial solution which is the set of all testable paths in the circuit. Due to the mutually exclusive nature of the critical events, it is known that

$$q(\Pi) = \sum_{p \in \Pi} \lambda_{\Omega_t}(p) \tag{5.5}$$

if any two paths in the circuit are not exactly identical. This is true in most practical applications [42]. Thus if we can compute λ_{Ω_t} efficiently, selecting top k paths with highest λ_{Ω_t} leads to the optimal solution. However, this is not possible unless we explicitly separate all testable paths from all paths in the circuit. In the following section, we develop an efficient algorithm to solve the optimization problem without the explicit enumeration. Even if we present the algorithm in the context of the testability, it can be used for the selection of paths with any specific property (e.g., functionally sensitizable, robustly testable, etc.).

5.4 Criticality Based Testable Path Selection Algorithm

5.4.1 Properties of Criticality

In this subsection, we derive some basic properties of the criticality in a path set.

Proposition 3. *Let U and V be path sets such that $U \subset V$. Then, $\lambda_U(p) \geq \lambda_V(p)$ for all $p \in U$.*

Proposition 4. *Let U and V be path set such that $U \subset V$. Then, $\lambda_V(p) \geq \lambda_U(p) - P(S_{V \setminus U} < S_U)$ for all $p \in U$.*

To explain the motivation of these two propositions, we will consider an example with U=$\{p_1,p_2,p_3\}$ and V=$\{p_1,p_2,p_3,p_4,p_5\}$. Suppose that U was of interest initially, so we computed λ_U. Later we have found p_4 and p_5, so we may need to compute λ_V for every path. In this case, the two propositions allow us to estimate λ_V efficiently without the re-computation. Figure 5.2 illustrates the change of the criticality. Since additional paths are introduced,

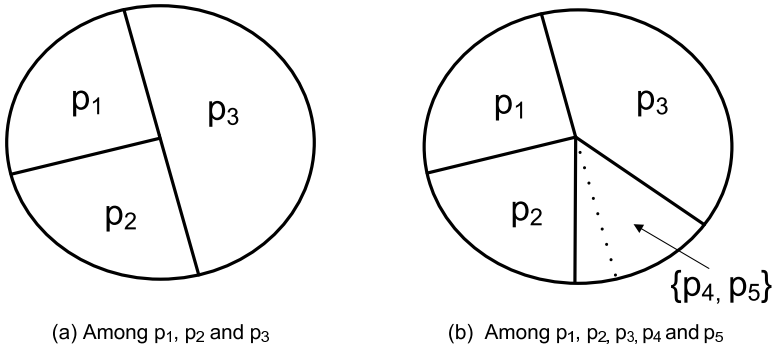


Figure 5.2: The regions that each path is critical in the process parameter space are shown. The area of each region represents the criticality. For example, $\lambda_U(p_1) = \lambda_V(p_1) = 0.25$.

more conditions are required for each path to be critical, so the criticality of each path decreases (Proposition 1). However, the decrement is at most the probability that the minimum slack of new paths is less than that of existing

paths (Proposition 2). Thus, we have an upper bound and a lower bound of the new criticality when additional paths are introduced or, conversely, some paths are eliminated.

Proposition 5. *Let U', U, V' and V are path sets such that $U' \subset U$ and $V' \subset V$. Then, $P(S_{U'} < S_V) \leq P(S_U < S_{V'})$.*

This proposition can also be proved by inclusion relation in the sample space.

Proposition 6. *In a path set U, the number of the paths whose criticality in U is greater than τ is at most $1/\tau$.*

Proof. Let k be the number of paths whose $\lambda_U > \tau$. Suppose $k > 1/\tau$. Then the sum of the criticality of the k paths is greater than 1 which is a contradiction. □

5.4.2 Proposed Algorithm

Our algorithm takes a timing graph G and the number of paths to be selected, k, as input and lists k testable paths that maximize TCM. In the beginning, we perform statistical static timing analysis on the timing graph and thus arrival times, required arrival times and slacks are available for each vertex. These timing quantities are represented in the form of (1.1) (the canonical form). Basically, we will adopt the branch and bound framework proposed in [89]. The framework uses recursive depth-first traversal with efficient pruning. From each end point (input pins of flops and primary outputs) in the

timing graph, the traversal begins toward start points (output pins of flops and primary inputs). The framework maintains the k best paths found so far during the traversal and whenever a new path is found, one path among the k paths is replaced, or the new path is just discarded depending on a chosen test quality metric. Figure 5.3 illustrates the recursive depth-first traversal

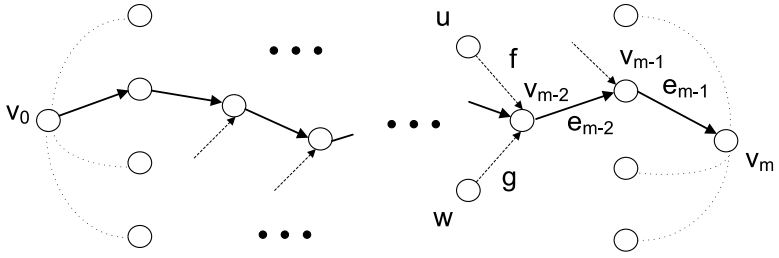


Figure 5.3: A recursive depth-first traversal

which is visiting the vertex h via the sub-path π and will search the solution space that consists of the paths going through π. In the vertex h, the solution space is naturally split into the paths via f and π and the paths via g and π (branching). Suppose that it visits f first. Then the sub-path π becomes (f, h, i, j, k), and the depth-first traversal will search the sub-space that consists of the paths via (f, h, i, j, k). Before searching this sub-space, we can efficiently check if it is worthwhile to search the sub-space using the following two strategies.

- If a sub-path is not testable, all the paths going through the sub-path are not testable. Thus we check if the sub-path π is testable using an incremental SAT solver [35] or RESIST [27], and if it is not testable, the branch is pruned.

- The slack of the sub-path π represents the minimum of the slacks of all the paths via the sub-path π. Using the sub-path slack, we can see if there exists a better path in the sub-space than current k best paths. The detailed condition will be explained later.

The first condition allows us to consider testable paths only, and due to the second condition, our algorithm inspects only a small portion of the testable paths.

Algorithm 8 SelectTestableCriticalPaths

Input: G, k; m

1: $\Pi \leftarrow \emptyset$, $\Sigma \leftarrow \emptyset$
2: **for** each testable path p obtained from DFS of G with pruning **do**
3: **if** $|\Pi| < k$ **then**
4: Insert p to Π and continue
5: **end if**
6: Insert p to Σ
7: **if** $|\Sigma| < m$ **then**
8: continue
9: **end if**
10: $\Psi \leftarrow \Pi \cup \Sigma$
11: **for** each path $s \in \Psi$ **do**
12: compute the path slack $S(s)$
13: compute the complement slack $\overline{S}(s)$
14: $\lambda_{\Psi}(s) = P(S(s) \leq \overline{S}(s))$
15: **end for**
16: update Π with top k paths with highest λ_{Ψ}
17: $\Sigma \leftarrow \emptyset$
18: **end for**
19: **return** Π

The overall algorithm is shown in Algorithm 8. Our algorithm maintains k best paths at any given time and the set of the k best paths is denoted

by Π (line 1). In [89], whenever a new path p is found, the path p is examined and Π is updated if desirable. However, our algorithm collects m paths before updating Π (line 7). Once m paths are accumulated, the algorithm considers the replacement of the paths in Π with the paths in Σ. In other words, $\Psi = \Pi \cup \Sigma$ becomes a set of candidate paths (line 10), and we find k best paths again out of Ψ. Note that the size of Ψ is $k + m$. We compute the criticality among the candidate paths (i.e., λ_Ψ). To compute λ_Ψ, for each path in Ψ, we first calculate the *complement slack* (line 13), which is the minimum of the slacks of all the other paths in Ψ except the path. Then, λ_Ψ is efficiently computed by (1.2) (line 14). Finally, the best k paths become top k paths with highest λ_Ψ (line 16).

Due to Proposition 3, the criticality λ_Ψ becomes an upper bound of λ_{Ω_t} and paths with high λ_Ψ can be considered as promising paths. Proposition 3 also implies that the larger m we use, the tighter the bound is. Thus, a large m value can improve TCM compared to the case $m = 1$. Actually, if we set m such that $m - k$ is the number of the total testable paths in the circuit, $\lambda_\Psi = \lambda_{\Omega_t}$ and the algorithm produces the optimal solution.

A straightforward algorithm to compute the complement slacks requires $O((m + k)^2)$ time, while the *binary partition tree* proposed in [76] reduces it to $O(m + k)$ time. Given the candidate path set Ψ, we can construct a balanced binary tree where each leaf corresponds to a path, as in Figure 5.4. We will compute the slack and the complement slack for each node in the tree. The slack of each leaf is set to the slack of the corresponding path, and

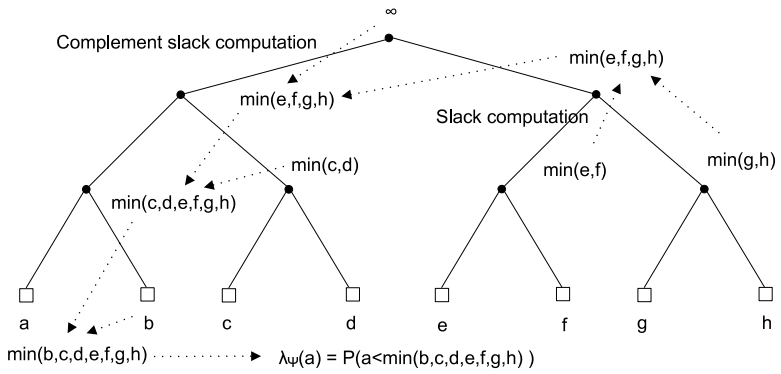


Figure 5.4: A binary partition tree to compute λ_Ψ

the complement slack of the root is initialized to infinity. Then, the slack and complement slack of the other nodes are calculated as follows. First, we traverse the binary tree in the bottom-up fashion, computing the minimum of two child for each node. This becomes the slack of the node. Once we reach the root, we traverse the binary tree again in the top-down fashion, computing the minimum of the parent complement slack and the sibling slack. This becomes the complement slack of each node, and the complement slack of each leaf becomes the complement slack of the corresponding path. In this way, the complement slacks are computed and then the criticality values are calculated. Finding the kth largest value in a list can be done in a liner time using the *median of medians* algorithm (a liner time selection algorithm) which also gives us the top k paths with highest criticality. Thus, the overall complexity of updating Π is $O(k+m)$. Let N be the number of the inspected paths (not pruned paths). Then, the update of Π is performed N/m times, and the time complexity of our algorithm is $O((k+m)N/m)$. Therefore, if we

set $m = O(k)$, the time complexity becomes $O(N)$, and the space complexity becomes $O(k)$.

5.4.3 Pruning Methods

During the depth first traversal, a sub-path π and its slack are given and we find out if there exists a better path in the paths via π than the current k best paths Π. At any given time, Ψ (previous candidate paths including Π), Π (k best paths), Σ (newly found paths) and λ_Ψ for each path in Ψ are available and they can be used for pruning.

First, we formally define the condition that a candidate path is discarded in the proposed algorithm when $m = 1$.

Definition 11. *For a path p, if $\lambda_{\Pi\cup\{p\}}(p) \leq \lambda_{\Pi\cup\{p\}}(s)$ for all $s \in \Pi$, then the path p is worthless.*

Let Ω_π be denote the set of all paths via π. A straightforward condition to prune π is as follows.

Proposition 7. *If $P(S(\pi) < S_\Pi) \leq \lambda_{\Pi\cup\Omega_\pi}(s)$ for all $s \in \Pi$, then all paths via π are worthless.*

Proof. We will find an upper bound of the left hand side in the inequality of Definition 11 and an lower bound of the right hand side which will give us a sufficient condition. For any path p in Ω_π, we have an upper bound

$$\lambda_{\Pi\cup\{p\}}(p) = P(S(p) < S_\Pi) \geq P(S(\pi) < S_\Pi) \tag{5.6}$$

by Proposition 5. For every $s \in \Pi, p \in \Omega_\pi$, we have a lower bound

$$\lambda_{\Pi \cup \{p\}}(s) \geq \lambda_{\Pi \cup \Omega_\pi}(s) \tag{5.7}$$

by Proposition 3. □

However, this condition requires computing $\lambda_{\Pi \cup \Omega_\pi}$ for all $s \in \Pi$ which means that we need to traverse the binary partition tree at every branching point in the depth-first traversal. Clearly, this is time consuming so we may desire to perform pruning in a constant time using already computed criticality λ_Ψ.

Proposition 8. *If* $2 \times P(S(\pi) < S_\Pi) \leq min_{s \in \Pi} \lambda_\Psi(s)$*, then all paths via* π *are worthless.*

Proof. We can find a looser bound than the lower bound used in Lemma 7. For every $s \in \Pi, p \in \Omega_\pi$, we have a lower bound

$$\begin{aligned} \lambda_{\Pi \cup \{p\}}(s) \geq \lambda_{\Pi \cup \Omega_\pi}(s) &\geq \lambda_{\Psi \cup \Omega_\pi}(s) \\ &\geq \lambda_\Psi(s) - P(S(\pi) < S_\Psi) \end{aligned} \tag{5.8}$$

by Proposition 4. Thus,

$$\begin{aligned} & P(S(\pi) < S_\Pi) \leq \lambda_\Psi(s) - P(S(\pi) < S_\Psi) \\ \Leftrightarrow\ & P(S(\pi) < S_\Pi) + P(S(\pi) < S_\Psi) \leq \lambda_\Psi(s) \end{aligned} \tag{5.9}$$

is a sufficient condition to the inequality of Definition 11, and $2 \times P(S(\pi) < S_\Pi) \leq \lambda_\Psi(s)$ becomes a more stringent condition. □

These exact methods allow us to prune fruitless branches effectively when m is a small value. However, as m increases, we update Π in a lazy manner and this can degrade the performance of the pruning. In other words, Σ may have some paths that are useful for pruning but they are not used properly when m is large. Since it is important to use a sufficiently large m value for both runtime and quality of results, our algorithm employs a heuristic method, which is to compare the paths via π to the path with the minimum λ_Ψ in Π. If all paths in the branch π are worse than the worst path in Π, the branch is highly likely to be fruitless. Let w be the path with the minimum λ_Ψ in Π. For all paths p via π, we have

$$\begin{aligned} \lambda_{\Psi\cup\Sigma\cup\Omega_\pi}(p) =& P(S(p) < min(S_{\Omega_\pi\setminus\{p\}}, S_\Psi, S_\Sigma)) \\ &\leq P(S(\pi) < min(S_\Psi, S_\Sigma)) \end{aligned} \tag{5.10}$$

by Proposition 5. Thus, if

$$P(S(\pi) < min(S_\Psi, S_\Sigma)) \leq \lambda_{\Psi\cup\Sigma\cup\Omega_\pi}(w) \tag{5.11}$$

is satisfied, all paths via π have a lower criticality than w, and we consider that the paths via π are worthless.

5.5 Selection by a Threshold

The proposed algorithm in the previous section is stable in the sense that before running the algorithm, we can estimate the amount of the result (i.e., the number of selected paths) and the run time, which is usually proposition to the number of paths. Thus, users seem comfortable to select a

desired number of paths. However, the goal of testing is to assure a desired quality all the time so we often question how many paths we should select to obtain a desired TCM value. This is a difficult problem, but in most cases, we want to cover all potentially critical paths, so it is useful to select all paths whose criticality is greater than a threshold value. Deterministic STA tools also provide similar two options, and in this section we present a modification for supporting the selection-by-a-threshold.

Our modified algorithm takes a timing graph and a threshold value τ as input and tries to list testable paths whose criticality among testable paths is greater than or equal to the threshold value τ. The number of paths to be selected is automatically determined depending on the amount of variation and the circuit. The overall algorithm is shown in Algorithm 9. Basically the

Algorithm 9 SelectTestableCriticalPathsByThreshold

Input: G, τ

```
1: Π ← ∅, Ψ ← ∅
2: for each testable path p obtained from DFS of G with pruning do
3:    add p to Π
4:    if a certain criterion is satisfied then
5:       for each path p in Π do
6:          let Ψ be the set of all paths found so far from the DFS
7:          compute λ_Ψ(p)
8:          if λ_Ψ(p) < τ then
9:             remove p from Π
10:         end if
11:      end for
12:   end if
13: end for
14: return Π
```

algorithm accumulates found testable paths into Π (line 3), and if a certain criterion is satisfied (line 4), we examine Π and discard unnecessary paths. We call the criterion the *examination criterion*, which will be explained later. If λ_{Ω_t} of a path is smaller than τ, the path is not necessary and can be discarded. However, as mentioned earlier, λ_{Ω_t} is difficult to calculate so we use an upper bound provided by Proposition 3. Let Ψ be the set of all paths found so far from the DFS (line 6). For each path in Π, we compute λ_Ψ instead of λ_Π because λ_Ψ is a tighter upper bound of λ_{Ω_t} than λ_Π. If the upper bound is less than τ, we can safely discard the path (line 8 and 9). Note that to compute λ_Ψ, we do not have to maintain actual paths in Ψ, and we can just store the minimum slack of discarded paths. In the case that the pruning is not performed and the examination is done only once in the end, $\Psi = \Omega_t$ and Algorithm 9 produces the exact result. If the pruning is performed or the examination is done in the middle, Ψ is a subset of Ω_t and, since we use an upper bound to discard paths, the algorithm results in a superset of the exact solution. However, if τ is small enough, the pruned paths $(\Omega_t - \Psi)$ have negligible impact on the path criticality (i.e, $\lambda_\Psi \approx \lambda_{\Omega_t}$). Also, in our application, the exact set is not necessary, and the superset can be a set of good candidate paths for at-speed test. After we perform the examination, we hope to eliminate some paths from Π in order to keep a decent memory usage. One may think that the number of eliminated paths is not guaranteed so it is not possible to reduce the memory usage. However, Proposition 6 allows us to calculate how many paths will remain after the examination. After each

examination, the size of Π is at most $1/\tau$.

The frequency of the examination determines the time and space complexity of the algorithm. We can consider two extreme examination criteria; one is the case that the examination is performed only once in the end, and the other is the case that it is performed whenever a new path is founded. Both the time and space complexity in the first case are $O(N)$ where N is the number of the testable paths found by the DFS. In the second case, the time complexity is $O(min(N, 1/\tau)N)$, and the space complexity is $O(min(N, 1/\tau))$.

5.5.1 Modification of the Pruning Method

For the selection by a threshold, the pruning method should also be modified. The heuristic pruning fits for Algorithm 8, whereas an exact pruning method can be employed to Algorithm 9. We maintain S_Ψ at any time, and we are given a subpath π and its slack during the depth first traversal. For each branch, if

$$P(S(\pi) \leq S_\Psi) \leq \tau \tag{5.12}$$

then, for every path p passing though π,

$$\lambda_{\Omega_t}(p) = P(S(p) \leq S_{\Omega_t}) \leq P(S(\pi) \leq S_\Psi) \leq \tau \tag{5.13}$$

due to Proposition 5 which means that all paths passing through π have smaller criticality values than τ so we can prune the branch. Due to this pruning, the number of discovered (and inspected) paths from the DFS is much smaller than the total number of testable paths, and Algorithm 9 can be run very efficiently.

5.5.2 Incrementality

For path selelction tools, users often request more paths than the already selected paths. In the traditional flow, this is mainly because some selected paths are determined as untestable in the ATPG process and the test coverage loss occurs. In our proposed flow, this will not happen so the incrementality that speeds up the algorithm using the results of the previous runs may not be as important as the traditional flow. However, there are still some needs for the incrementality. For example, the test budget may increase so more paths can become affordable. If some testable paths are available from the previous runs, we can easily speed up our algorithm.

The upper bound λ_Ψ becomes tight as the size of Ψ increases. Also it is more effective if Ψ contains many paths with high criticality. However, it takes some time for the bounding to become effective by finding several testable paths with high criticality. If the algorithm takes Π and S_Ψ over from a previous run, the bounding is very effective from the beginning. This bounding can prune the paths that are already in Π, so the algorithm should start with Π, and only newly discovered paths should be added to Π. Since the incremental algorithm prunes more testable paths from the beginning, the computed criticality becomes a lessen upper bound of λ_{Ω_t} than that when the incrementality is disabled, and the produced path set is a superset of the result of the non-incremental algorithm.

5.6 Integration of a SAT Solver

Boolean satisfiability is a problem of finding an assignment for making a given boolean formula evaluate to true, or proving that such an assignment does not exist (*unsatisfiable*). A *literal* is a variable or the negation of a variable, and a *clause* is disjunctions of literals. We are typically interested in the special cases of boolean satisfiability where the formulae are required to be in the *conjunctive normal form* (i.e., conjunctions of clauses). To check the testability, we first take a circuit as input and assign a variable to each net. Then we extract clauses for enforcing the relations among the variables due to the function of each gate. The derivation process and more examples are found in [35, 39]. As mentioned earlier, to prune untestable branches, we add some clauses and solve the current satisfiability problem whenever the DFS goes forward. In other words, we solve the testability problem of a path incrementally to find out untestable branches. If a SAT solver used supports this type of usage (i.e., an incremental SAT solver is used), the approach can be efficient [35]. However, there is also some penalty in this approach. For example, if all paths are testable, the number of SAT calls increase unnecessarily. In the case of [35], the SAT solver seems specially optimized for this scenario, and each SAT call seems very light-weighted. However, this is not the case in general SAT solvers.

To deal with this issue, we propose an alternative approach that leverages the *conflict analysis*, with which modern SAT solvers are equipped. Without checking the testability of subpaths, we only check the testability of a full

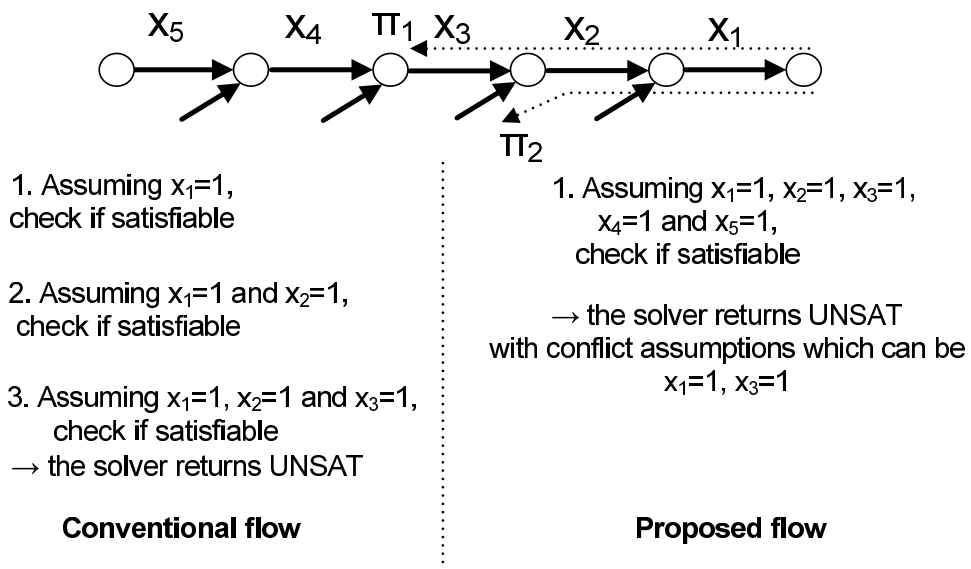


Figure 5.5: An incremental solver has been used to identify untestable branches. If the conflict analysis is used, we can locate them more efficiently even without using incremental SAT.

path. If the path is not testable, the SAT solver performs the final conflict analysis which will imply which segments cannot satisfy the constraints for the testability at the same time. In our implementation, we assign one variable for each segment of a path, and the variable becomes true if the corresponding segment satisfies the testability constraint. Then we let the SAT solver finds an assignment that all the variables become true. In the case of *minisat* [21], it takes a set of variables called the *assumptions* and find an assignment that makes all assumptions evaluate to true. Thus we put the variables that represent the testability of each segment into a list of assumptions. If it is unsatisfiable under the assumptions, the minisat will produce a set of conflict assumptions, by which we can locate the untestable branch, from which we

can start over the DFS.

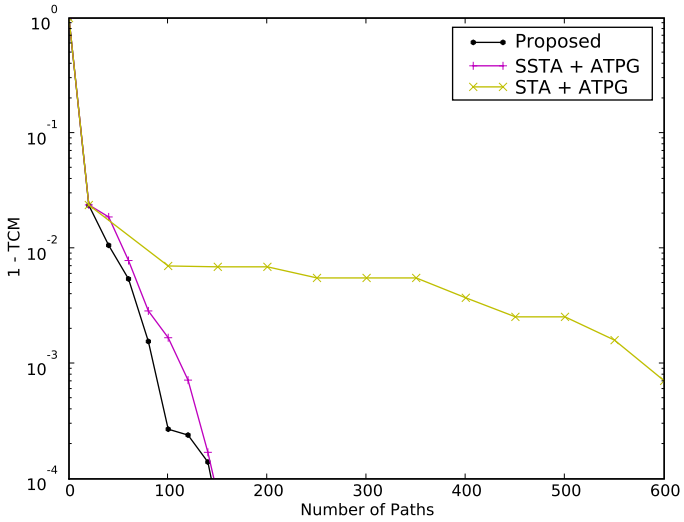
Figure 5.5 illustrates the conventional flow and the proposed flow, and a part of the DFS tree is shown on the top. The variables for the testability of each segment are denoted by $x_1, x_2, \cdots, x_5$. Thus if all the variables can be set to true at the same time, the path can be testable. Let us suppose that π_1 is an untestable branch (i.e., all paths via π_1 are not testable) so we want to go into π_2 by pruning π_1. In the conventional flow, we incrementally check the testability and in this case, we found π_1 is an untestable branch after 3 calls. In the proposed flow, we just check the testability of the full path, and the SAT solver will indicate the path is untestable and will produce the conflict assumptions. Even if π_1 is the shortest untestable branch (i.e., the branch corresponding to x_1 and x_2 is testable), there are several possible explanations. For example, the conflict assumptions can be $x_1 = 1$ and $x_3 = 1$ which means that the corresponding segments cannot satisfy the constraints together. Similarly, they can be $x_2 = 1$ and $x_3 = 1$. However, if π_1 is the shortest untestable branch, the conflict assumptions should include $x_3 = 1$ but $x_4 = 1$ and $x_5 = 1$. Thus, we can locate the shortest untestable branch easily by finding the leading conflict assumption and can go into π_2 after just one SAT call.

We may enhance the proposed approach further. Let us consider the case that $x_2 = 1$ and $x_3 = 1$ are the conflict assumptions and let us denote the segments corresponding to $x_1, \cdots, x_5$ by $s_1, \cdots, s_5$, respectively. There may be other paths passing through s_2 and s_3 but s_1, and the SAT solver is called

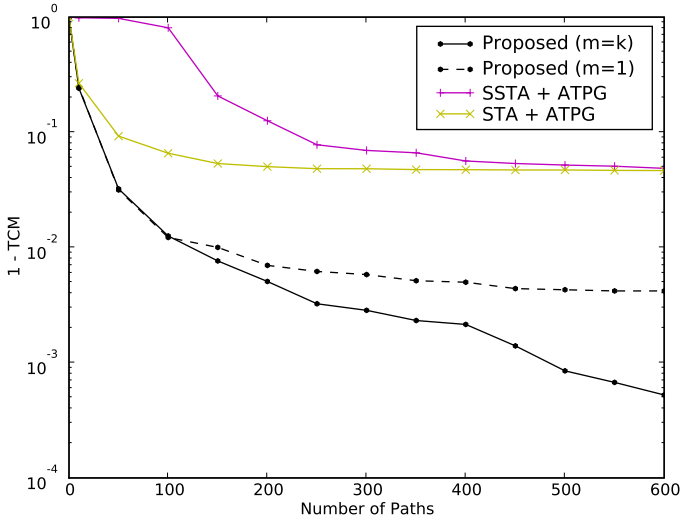
to see if such paths are testable. However, we already know that those paths are not testable. Thus if that information is stored, we can reduce the number of SAT calls more. However, some SAT solvers internally store the information by adding a clause called the *learnt clause*. In this case, the benefit of storing it externally may be marginal.

5.7 Experimental Results

We implemented the proposed algorithms in C++, and used ISCAS85 circuits as benchmarks. We also used an ethernet IP core (eth), the tv80 microprocessor core (tv80), and the or1200 open-source microprocessor including a 32-bit, 5-stage Wallace multiplier. To check the testability, *minisat* [21] was incorporated in our implementation. All experiments were performed on a 3.0GHz 2 Xeon X5570 Linux machine. The tv80 and eth are technology-mapped to the IBM 45nm library, and or1200 and ISCAS85 circuits use the TSMC 180nm library. The nominal delay annotated to each edge in the timing graph is the sum of the gate and wire delays obtained from the Standard Delay Format (SDF) file. In this experiment, a quad tree with 3 levels is used to model spatial correlation which results in 21 global sources of variation. This allows us to verify the proposed algorithms under difficult conditions. The global sources of variation associated with the first, the second and the third level can change the delay of each edge up to $\pm 4\%$, $\pm 5\%$ and $\pm 6\%$ of its nominal value in the 3σ case, respectively. The delay of each edge also has the independent variation, and the 3σ point is 5% of the nominal delay. In our



(a) eth (avg. path length: 13.1 gates, testable ratio: 82.4%)



(b) tv80 (avg. path length: 37.6 gates, testable ratio: 0.03%)

Figure 5.6: The optimality in pre-ATPG path selection achieved by SSTA is destroyed during the ATPG process. Our testability driven approach considers the testability in the first place and achieves superior quality of results even in the extremely low testable ratio.

experiments, we assume that non-robust tests can determine the delay of the path being targeted irrespective of other path delays as in robust tests. We first compare Algorithm 8 with two other methods. One method (SSTA+ATPG) is to select paths with highest criticality λ_Ω using SSTA without considering the testability and then to check the testability using ATPG. The untestable paths are discarded and the procedure is iterated until a desired number of paths are obtained. The other method (STA+ATPG) is the same as SSTA+ATPG but nominal slacks (nominal delay) are used instead of the path criticality. In this way, given a desired number of paths, we can construct three different path sets from each method, and the (critical) testable path coverage (TCM) is measured by Monte Carlo simulation using 40000 samples. Note that only testable paths are used in measuring TCM.

Figure 5.6 shows TCM archived by each method as the desired number of paths increases. From SSTA+ATPG method, we obtain the *testable ratio*, which is the ratio of testable paths to the total selected paths. According to our experiments, the lengths of paths (in the number of gates) shows a strong correlation with the testable ratio, so we also show the average path length of the total selected paths from SSTA+ATPG method. The correlation seems to come from the fact that longer paths require more constraints, making the SAT problem for ATPG more difficult to be satisfied. In eth, many paths are testable, and SSTA+ATPG provides significantly better quality than STA+ATPG. On the other hand, the quality is severely degraded in tv80 due to the very low testable ratio. The proposed algorithm produces a high quality

path set regardless of the testable ratio. Figure 5.6(b) also show the improvement of TCM by a large m value, with which the proposed algorithm finds a solution in a less "greedy" manner.

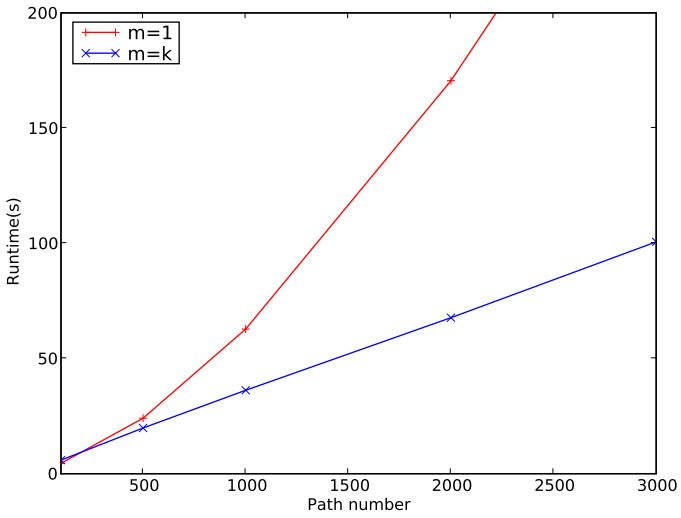


Figure 5.7: The pruning technique allows us to select paths efficiently, and the runtime grows linearly with respect to the number of selected paths when $m = k$.

To show the runtime scalability of the proposed method with respect to the number of selected paths, we selected paths for each case that $m = 1$ and $m = k$. For this experiment, the circuit or1200 is used which consists of about 40k gates. Figure 5.7 shows that the runtime increases slowly when $m = k$, compared to the case $m = 1$. Table 5.1 compares Algorithm 8 with SSTA+ATPG in ISCAS85 circuits. Both methods are iterated until 5 paths are obtained. However, for c499 and c1355, 50 paths are selected because they have many near-critical paths. In SSTA+ATPG, the threshold value

Table 5.1: Our experiment for SSTA+ATPG is performed in a single tool and the runtime results are much better than that in the typical industrial setting where the two tools are separated. Nevertheless, the proposed method shows the significant speed-up over SSTA+ATPG. Note that the proposed method also generates test patterns.

Circuit	#paths	SSTA+ATPG						Proposed		
		ratio	#i	TCM	CPU(s)			#i	TCM	CPU(s)
					ATPG	PathSel	Total			
c17	18	100	5	1.000	0.000	0.000	0.000	1	1.000	0.000
c432	119332	2.8	5	0.373	0.077	0.003	0.082	1	0.780	0.019
c499	332928	45.8	1	0.888	0.016	0.013	0.030	1	0.901	0.079
c880	14470	100	1	0.913	0.010	0.002	0.010	1	0.903	0.005
c1355	332928	49.2	1	0.863	0.017	0.011	0.028	1	0.870	0.058
c1908	1867970	19.5	322	0.0	62.1	395.8	464.9	1	0.869	0.238
c2670	54214	83.0	1	0.870	0.017	0.011	0.029	1	0.870	0.030
c3540	5892630	0.3	15	0.0	0.673	0.066	0.747	1	1.000	0.134
c5315	644060	26.7	2	1.000	0.047	0.012	0.061	1	1.000	0.056
c6288	3.7269×10^{16}	4.8e-7	14	0.006	17780.0	442.3	18369.2	1	0.991	33.16
c7552	611804	2.0	12	1.000	0.490	0.015	0.507	1	1.000	0.063
avg.				0.629	1784.3	83.8	1712.3		0.926	3.077

begins with 0.001 and is multiplied by 0.1 for each iteration. Table 5.1 shows the testable ratio (ratio), the number of the iteration (#i), TCM and the runtime. The circuits c432, c1908, c3540, c6288 and c7552 have a small number of testable paths similar to tv80, and both the runtime and the quality are poor. The proposed method can easily handle even c6288 that is notorious for numerous paths. For a given number of paths, the proposed method produces a better path set than that from SSTA+ATPG, but it is difficult to determine an adequate number of paths for achieving a high quality. For example, if we desire to achieve more than 80% TCM, 5 paths are inadequate in c432.

Alternatively, we can select paths by a threshold τ. For ISCAS85 circuits, Algorithm 9 selected paths using two threshold values, $\tau = 10^{-4}$ and $\tau = 10^{-5}$. Then, we ran Monte Carlo simulations with 100,000 samples to obtain TCM. Also during the Monte Carlo simulations, we counted the number of testable paths that became critical among testable paths at least once. For convenience, we will just call such paths critical paths. Table 5.2 shows the number of critical paths (# critical), the number of selected paths (# sel. paths), TCM and the runtime. The proposed algorithm chose the number of paths to be selected based on the circuit and the amount of variation. Since c499 and c1355 have many-near critical paths, it selected adequate numbers of paths from them. Readers can identify a good correlation between the number of critical paths and the number of selected paths. Since we inspected 10^6 samples in this experiment, the threshold 10^{-4} is not sufficient to capture all the critical paths, and the TCM in c499 is not 100%. However, with the

Table 5.2: If 100% TCM is desired, we can select paths by a threhold value. Algorithm 9 with a low threshold value as input found all paths that are testable and can potentially have the least slack among all testable paths.

		$\tau = 10^{-4}$			$\tau = 10^{-5}$		
Circuit	# critical	# sel. paths	TCM	CPU(s)	# sel. paths	TCM	CPU(s)
c17	2	4	1.000	0.002	4	1.000	0.005
c432	30	36	1.000	0.022	39	1.000	0.024
c499	94	131	0.997	0.046	207	1.000	0.061
c880	7	10	1.000	0.006	11	1.000	0.005
c1355	102	148	1.000	0.042	262	1.000	0.051
c1908	10	20	1.000	0.224	29	1.000	0.218
c2670	31	58	1.000	0.038	65	1.000	0.041
c3540	3	4	1.000	0.147	4	1.000	0.171
c5315	4	19	1.000	0.048	44	1.000	0.056
c6288	16	86	1.000	76.017	159	1.000	103.717
c7552	2	7	1.000	0.057	8	1.000	0.054

threshold 10^{-5}, we successfully captured all the critical paths, although more paths are used.

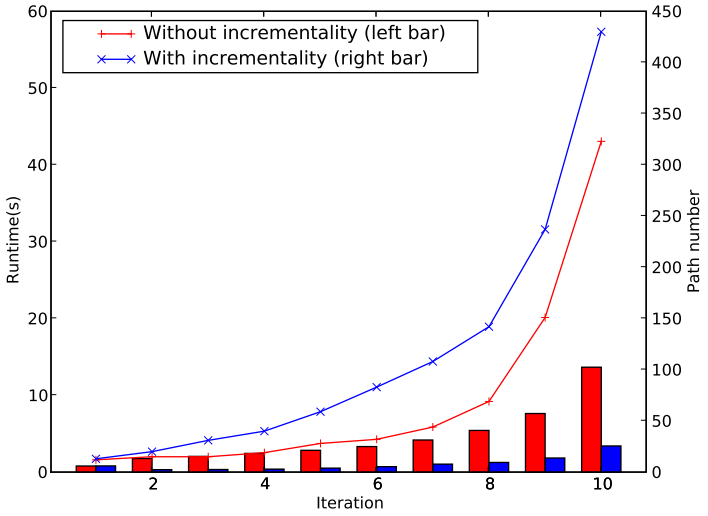


Figure 5.8: If the results of previous runs are available, we can further speeds up the algorithm.

The incrementality is supported in the selection-by-a-threhold. To show the benefit of the incrementality, we selected paths by several iterations while decreasing the threshold. The threshold is decreased by 0.001 from 0.01. Figure 5.8 shows the runtime values when the incrementality is enabled and disabled, respectively. If the incrementality is enabled, the runtime becomes faster, but the result path set gets bigger for the reason explained in the incrementality section.

Table 5.3 compares the proposed SAT integration method with the one using incremental SAT. As discussed earlier, the incremental method suffers

Table 5.3: Our novel SAT Integration method can enhance the performance substantially.

	Incremental Method		Proposed		
Circuit	# SAT calls	CPU(s)	# SAT calls	CPU(s)	Improv.
c17	28	0.002	8	0.000	5x
c432	2768	0.050	975	0.019	2.63x
c499	6037	0.326	1415	0.079	4.12x
c880	176	0.007	36	0.005	1.4x
c1355	4681	0.242	896	0.058	4.17x
c1908	13501	0.539	4884	0.238	2.26x
c2670	1923	0.064	636	0.030	2.13x
c3540	6498	0.753	1627	0.134	5.62x
c5315	3153	0.302	357	0.056	5.39x
c6288	396567	352.394	32347	33.16	10.63x
c7552	4925	0.338	719	0.063	5.37x
avg.	40023	32.274	3991	3.077	4.43x

from a large number of SAT calls, while the proposed method reduces the number of SAT calls significantly which in turn decreases the runtime substantially. The average speed-up is about 4.43X which is achieved without any compromise.

5.8 Conclusions

We have presented a testability driven approach in criticality-based path selection with post-ATPG results evaluated by Monte Carlo simulation. In deterministic path selection, this approach is considered as an option for speed-up, but our results have shown that it is necessary to ensure high test quality in statistical path selection. Since the testability check using a SAT

solver actually involves test generation, our algorithm not only produces k testable paths but also generates the test patterns sensitizing them in a single run. If the test patterns are run a tester, the benefits of statistical methodology will be demonstrated on a silicon.

Chapter 6

Conclusions

This dissertation aims to develop variation-aware design automation and testing techniques. Our efforts for achieving this goal range from fundamental theories and computational methods to high-level applications. Our target applications leverage block-based SSTA, which is very fast but suffers from poor accuracy. The combination of our two fundamental techniques, refactoring and the conditioning operation, raises the accuracy of block-based SSTA to Monte Carlo simulation level only using analytical and algorithmic methods. As demonstrated in criticality computation, it can enhance various existing SSTA applications immediately. The enhanced scalability in comparison with Monte Carlo simulation used to be uncharted territory, and we strongly believe that it can foster new SSTA applications in electronic design automation and VLSI testing. Some future directions may include:

- Refactoring is a very general technique that relies only on one algebraic property and works in a highly abstracted domain. The issues being dealt with in refactoring are closely related to the reason why many efficient algorithms in graphs cannot produce the optimal solutions. We believe that further studies on refactoring are beneficial to many other

areas even beyond EDA.

- In the conditioning operation, the random variables representing the parameters become correlated due to given conditions. Computing the conditional covariance takes up a large part of the runtime. However, the benefit of explicitly computing it is not fully justified. If assuming their independence does not degrade the accuracy severely, we may achieve a significant speed-up at the cost of marginal accuracy loss.

- Our proposed criticality computation methods can accurately locate where optimization is needed in the circuit. Thus using the methods, various yield optimization techniques such as gate sizing, buffer insertion and threshold voltage assignment can be developed. However, it should be noted that sensitivities are subtle, so coarse grain knobs such as discrete gate sizing may not be very effective. Currently, research on transistor sizing for custom blocks is ongoing and it seems appropriate because the knob is fine enough and the benefits of such optimization are proven already in analog circuit designs. In order to guarantee adequate yield in nanoscale technologies, it is required to find such fine knobs for synthesized digital blocks.

- Our proposed test synthesis techniques are quite mature, and they are ready to be proved on silicon. Thus, we believe that future research should be performed in design-for-test (DFT) rather than pre-silicon test synthesis. It is clear that the measurement on natural critical paths pro-

vides better correlation to the circuit performance than that of process ring oscillators (PSRO). However, the measurement on natural critical paths is still noisy with DFT circuits currently employed in industry. Thus, the methodologies for the measurement need to be improved.

Appendices

Appendix A

Proof of Theorem 3

We continue to use the notation introduced in Theorem 3. However, without loss of generality, we here assume that $\mu_3 = 0$ and $\mu_4 = 0$.

Lemma 3.

$$\int_m^\infty x\varphi(x)dx = \varphi(m)$$

Proof. This lemma is proved in [17]. □

Lemma 4.

$$\int_m^\infty x^2\varphi(x)dx = m\varphi(m) + \Phi(-m)$$

Proof. This lemma is proved in [17]. □

We define a random variable Z as

$$Z = X - Y. \tag{A.1}$$

Then, we can easily identity that $a = \sigma_z$ and $\alpha = \mu_z/\sigma_z$. We will first prove (3.30) in Theorem 3. The probability density function is denoted by

ψ. Using (A.1), we get

$$\begin{aligned} E[T|X>Y] &= E[T|Z>0] \\ &= \frac{1}{\Phi(\alpha)} \int_0^\infty \int_{-\infty}^\infty t\psi_{T,Z}(t,z)dtdz \\ &= \frac{1}{\Phi(\alpha)} \int_0^\infty \psi_Z(z)dz \int_{-\infty}^\infty t\psi_T(t|Z=z)dt. \end{aligned} \tag{A.2}$$

Note that $\Phi(\alpha) = P(X > Y)$. Using the fact that $E[T|Z=z] = cov[T,Z](z-\mu_z)/\sigma_t^2$, (A.2) is reduced to

$$\begin{aligned} &\frac{1}{\Phi(\alpha)} \int_0^\infty \psi_Z(z) cov[T,Z] \left(\frac{z-\mu_z}{\sigma_z^2}\right) dz \\ &= \frac{cov[T,Z]/\sigma_t}{\Phi(\alpha)} \int_0^\infty \left(\frac{z-\mu_z}{\sigma_z}\right) \psi_Z(z)dz \\ &= \frac{cov[T,Z]/\sigma_t}{\Phi(\alpha)} \int_{-\mu_z/\sigma_z}^\infty n\varphi(n)dn \\ &= \frac{cov[T,Z]/\sigma_t}{\Phi(\alpha)} \int_{-\alpha}^\infty n\varphi(n)dn. \end{aligned} \tag{A.3}$$

Applying Lemma 3 to (A.3) yields

$$\begin{aligned} E[T|X>Y] &= \left(\frac{\varphi(\alpha)}{\Phi(\alpha)}\right) cov[T,Z]/\sigma_z \\ &= \beta cov[T,Z]/a \end{aligned} \tag{A.4}$$

which becomes (3.30) in Theorem 3. Next, to obtain (3.31) in Theorem 3, we write

$$\begin{aligned} &E[TU|X>Y] \\ &= E[TU|Z>0] \\ &= \frac{1}{\Phi(\alpha)} \int_0^\infty \int_{-\infty}^\infty \int_{-\infty}^\infty tu\psi_{T,U,Z}(t,u,z)dtdudz \\ &= \frac{1}{\Phi(\alpha)} \int_0^\infty \psi_Z(z)dz \int_{-\infty}^\infty \int_{-\infty}^\infty tu\psi_{T,U}(t,u|Z=z)dtdu. \end{aligned} \tag{A.5}$$

Using the fact that $(T, U|Z = z) \sim \mathcal{N}(\overline{\mu}, \overline{\Sigma})$ where

$$\begin{aligned}\overline{\mu} &= \begin{bmatrix} cov[T,Z](z-\mu_z)/\sigma_z^2 \\ cov[U,Z](z-\mu_z)/\sigma_z^2 \end{bmatrix} = \begin{bmatrix} \overline{\mu}_1 \\ \overline{\mu}_2 \end{bmatrix} \\ \overline{\Sigma} &= \begin{bmatrix} \sigma_t^2 & cov[T,U] \\ cov[T,U] & \sigma_u^2 \end{bmatrix} \\ &\quad - \frac{1}{\sigma_z^2} \begin{bmatrix} cov[T,Z]^2 & cov[T,Z]cov[U,Z] \\ cov[T,Z]cov[U,Z] & cov[U,Z]^2 \end{bmatrix},\end{aligned}$$

(A.5) is reduced to

$$\begin{aligned}E[TU|Y > X] = \frac{1}{\Phi(\alpha)} \int_0^\infty (cov[T,U|Z=z] \\ + \overline{\mu}_1\overline{\mu}_2)\psi_Z(z)dz.\end{aligned} \tag{A.6}$$

Since the conditional covariance is independent of z, (A.6) can be re-written as

$$\begin{aligned}E[TU|Y > X] = cov[T,U|Z=z] \\ + \frac{1}{\Phi(\alpha)} \int_0^\infty \overline{\mu}_1\overline{\mu}_2\psi_Z(z)dz.\end{aligned} \tag{A.7}$$

By using the substitution $m = (z - \mu_z)/\sigma_z$ and applying Lemma 4, we obtain

$$\begin{aligned}&\int_0^\infty \overline{\mu}_1\overline{\mu}_2\psi_Z(z)dz \\ &= \frac{cov[T,Z]cov[U,Z]}{\sigma_z^2} \int_0^\infty (\frac{z-\mu_z}{\sigma_z})^2\psi_Z(z)dz \\ &= \frac{cov[T,Z]cov[U,Z]}{\sigma_z^2} \int_{-\alpha}^\infty m^2\varphi(m)dm \\ &= \frac{cov[T,Z]cov[U,Z]}{\sigma_z^2}(-\alpha\varphi(\alpha) + \Phi(\alpha)).\end{aligned} \tag{A.8}$$

Substituting (A.8) into (A.7) results in

$$\begin{aligned}&E[TU|Y > X] \\ &= cov[T,U] - (\alpha\varphi(\alpha)/\Phi(\alpha))cov[T,Z]cov[U,Z]/\sigma_z^2 \\ &= cov[T,U] - \alpha\beta cov[T,Z]cov[U,Z]/a^2.\end{aligned} \tag{A.9}$$

Finally, from (A.9) and (A.4), we obtain

$$\begin{aligned} & cov[TU|Y > X] \\ & \quad = E[TU|Y > X] - E[T|Y > X]E[U|Y > X] \\ & \quad = cov[T, U] - (\beta^2 + \alpha\beta)cov[T, Z]cov[U, Z]/a^2 \end{aligned} \tag{A.10}$$

which proves Theorem 3.

Bibliography

[1] OR1200 RISC processor. *http://www.opencores.org*.

[2] A. Agarwal, D. Blaauw, and V. Zolotov. Statistical timing analysis for intra-die process variations with spatial correlations. In *Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design*. IEEE Computer Society Washington, DC, USA, 2003.

[3] A. Agarwal, F. Dartu, and D. Blaauw. Statistical gate delay model considering multiple input switching. In *Proc. Design Automation Conf.*, pages 658–663, 2005.

[4] A. Agarwal, V. Zolotov, and D.T. Blaauw. Statistical timing analysis using bounds and selective enumeration. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, (9):1243–1260, 2003.

[5] J. Benkoski, E. Vanden Meesch, L. Claesen, and H. De Man. Efficient algorithms for solving the false path problem in timing verification. In *Proc. Int. Conf. on Computer Aided Design*, pages 44–47, 1987.

[6] S. Bhardwaj, S. Vrudhula, and D. Blaauw. τau: Timing analysis under uncertainty. In *Proc. Int. Conf. on Computer Aided Design*, pages 615–620, 2003.

[7] D. Blaauw, K. Chopra, A. Srivastava, and L. Scheffer. Statistical timing analysis: From basic principles to state of the art. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 27(4):589–607, 2008.

[8] R. K. Brayton, G.D. Hachtel, and A.L. Sangiovanni-Vincentelli. Multilevel logic synthesis. *Proc. of the IEEE*, (2):264–300, 1990.

[9] S. Browne and W. Whitt. Portfolio choice and the bayesian kelly criterion. *Advances in Applied Probability*, 28(4):1145–1176, 1996.

[10] H. Chang and S.S. Sapatnekar. Statistical timing analysis considering spatial correlations using a single PERT-like traversal. In *Proc. Int. Conf. on Computer Aided Design*, 2003.

[11] H. Chang, V. Zolotov, S. Narayan, and C. Visweswariah. Parameterized block-based statistical timing analysis with non-gaussian parameters, nonlinear delay functions. In *Proceedings of the 42nd annual Design Automation Conference*, page 76. ACM, 2005.

[12] H.C. Chen and DHC Du. Path sensitization in critical path problem. In *1991 IEEE International Conference on Computer-Aided Design, 1991. ICCAD-91. Digest of Technical Papers.*, pages 208–211, 1991.

[13] L.C. Chen, P. Dickinson, P. Dahlgren, S. Davidson, O. Caty, and K. Wu. Using transition test to understand timing behavior of logic circuits on UltraSPARC T2 family. In *Proc. Int. Test Conf.*, pages 1–10, 2009.

[14] K.T. Cheng and H.C. Chen. Delay testing for non-robust untestable circuits. In *Proc. Int. Test Conf.*, pages 954–961, 2002.

[15] J. Chung and J. A. Abraham. A hiearchy of subgraphs underlying a timing graph and its use in capturing topological correlation in ssta. In *Proc. Int. Conf. on Computer Aided Design*, pages 321–327, 2009.

[16] Jaeyong Chung and Jacob A. Abraham. Recursive Path Selection for Delay Fault Testing. In *Proc. VLSI Test Sympo.*, pages 65–70, 2009.

[17] C. E. Clark. The greatest of a finite set of random variables. *In Operations Research*, pages 85–91, 1961.

[18] B.D. Cory, R. Kapur, and B. Underwood. Speed binning with path delay test in 150-nm technology. *IEEE Design & Test of Computers*, 20(5):41–45, 2003.

[19] T.M. Cover and J. A. Thomas. *Elements of Information Theory*. New York: Wiley, 1991.

[20] A. Devgan and C. Kashyap. Block-based static timing analysis with uncertainty. In *Proc. Int. Conf. on Computer Aided Design*, pages 607–614, 2003.

[21] N. Eén and N. Sorensson. An extensible SAT-solver. In *Theory and Applications of Satisfiability Testing*, pages 333–336, 2004.

[22] T. Enami, S. Ninomiya, and M. Hashimoto. Statistical timing analysis considering spatially and temporally correlated dynamic power supply noise. In *Proc. Int. Symp. on Physical Design*, pages 160–167, 2008.

[23] T. Enami, S. Ninomiya, and M. Hashimoto. Statistical timing analysis considering spatially and temporally correlated dynamic power supply noise. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 28(4):541–553, 2009.

[24] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. Razor: a low power pipeline based on circuit level timing speculation. In *Proc. Int. Symp. Microarchitecture (MICRO-36)*, pages 7–18, 2003.

[25] K. Fuchs, F. Fink, and MH Schulz. Dynamite: an efficient automatic test pattern generation system forpath delay faults. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 10(10):1323–1335, 1991.

[26] K. Fuchs, M. Pabst, and T Rossel. Resist: a recursive test pattern generation algorithm for path delay faults considering various test classes. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 13(12):1550–1562, 1994.

[27] K. Fuchs, M. Pabst, and T Rossel. Resist: a recursive test pattern generation algorithm for path delay faults considering various test classes.

IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, 13(12):1550–1562, 1994.

[28] R. Gandikota, D. Blaauw, and D. Sylvester. Modeling crosstalk in statistical static timing analysis. In *Proc. Design Automation Conf.*, pages 974–979, 2009.

[29] K.R. Heloue, S. Onaissi, and F.N. Najm. Efficient block-based parameterized timing analysis covering all potentially critical paths. In *Proc. Int. Conf. on Computer Aided Design*, pages 173–180, 2008.

[30] V. Iyengar, T. Yokota, K. Yamada, T. Anemikos, B. Bassett, M. Degregorio, R. Farmer, G. Grise, M. Johnson, D. Milton, et al. At-speed structural test for high-performance ASICs. In *Proc. Int. Test Conf.*, pages 1–10, 2007.

[31] D. Josephson and B. Gottlieb. The crazy mixed up world of silicon debug. pages 665–670, 2004.

[32] C. Kashyap, P. Bastani, K. Killpack, and C. Amin. Silicon feedback to improve frequency of high-performance microprocessors: an overview. In *Proc. Int. Conf. on Computer Aided Design*, pages 778–782. IEEE Press, 2008.

[33] J.L. Kelly. A new interpretation of information rate. *IEEE Trans. on Information Theory*, 2(3):185–189, 1956.

[34] H. S. Kim and D. M. H. Walker. Statistical static timing analysis considering the impact of power supply noise in vlsi circuits. In *Proc. Int. Workshop on Microprocessor Test and Verification*, pages 76–82, 2006.

[35] J. Kim, J. Whittemore, JP Marques-Silva, and K Sakallah. On applying incremental satisfiability to delay fault testing. In *Proc. Design, Automation and Test in Eurpoe*, pages 380–384, 2000.

[36] K.S. Kim, S. Mitra, and P.G.Ryan. Delay defect characteristics and testing strategies. *IEEE Trans. on Design and Test of Computers*, 20(5):8–16, 2003.

[37] P.M. Kogge and H.S. Stone. A parallel algorithm for the efficient solution of a general class of recurrence equations. *IEEE Trans. on Computers*, 22(8):796–793, 1973.

[38] S.V. Kumar, C.V. Kashyap, and S.S. Sapatnekar. A framework for block-based timing sensitivity analysis. In *Proc. Design Automation Conf.*, pages 688–693, 2008.

[39] T. Larrabee. Test pattern generation using Boolean satisfiability. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 11(1):4–15, 2002.

[40] Jiayong Le, Xin Li, and L.T. Pileggi. Stac: statistical timing analysis with correlation. In *Proc. Design Automation Conf.*, pages 343– 348, 2004.

[41] W. Li, S. M. Reddy, and S. K. Sahni. On path selection in combinational logic circuits. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 8(1):56–63, 1989.

[42] X. Li, J. Le, M. Celik, and L.T. Pileggi. Defining statistical timing sensitivity for logic circuits with large-scale process and environmental variations. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 27(6):1041–1054, 2008.

[43] J.J. Liou, K.T. Cheng, S. Kundu, and A. Krstic. Fast statistical timing analysis by probabilistic event propagation. In *Proc. Design Automation Conf.*, pages 661–666. ACM, 2001.

[44] J.J. Liou, K.T. Cheng, and D. A. Mukherjee. Path selection for delay testing of deep-submicrometer devices using statistical performance sensitivity analysis. In *Proc. VLSI Test Sympo.*, pages 97–104, 2000.

[45] Q. Liu and S.S. Sapatnekar. Synthesizing a representative critical path for post-silicon delay prediction. In *Proc. Int. Symp. on Physical Design*, pages 183–190, 2009.

[46] Q. Liu and S.S. Sapatnekar. Synthesizing a representative critical path for post-silicon delay prediction. In *Proc. Int. Symp. on Physical Design*, pages 183–190, 2009.

[47] GM Luong and DMH Walker. Test generation for global delay faults. In *Proc. Int. Test Conf.*, 1996.

[48] GM Luong and DMH Walker. Test generation for global delay faults. In *Proc. Int. Test Conf.*, pages 433–442, 2002.

[49] H. Mahmoodi, S. Mukhopadhyay, and K. Roy. Estimation of delay variations due to random-dopant fluctuations in nanoscale cmos circuits. *IEEE J. Solid-State Circuits*, (9):1787– 1796, 2005.

[50] A.K. Makji, V.D. Agrawal, J. Jacob, and L.M. Patnaik. Line coverage of path delay faults. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 8(1):610–613, 2000.

[51] PC McGeer and RK Brayton. Efficient algorithms for computing the longest viable path in a combinational network. In *Proc. Design Automation Conf.*, pages 561–567, 1989.

[52] P.C. McGeer and R.K. Brayton. Efficient algorithms for computing the longest viable path in a combinational network. In *Proc. Design Automation Conf.*, pages 561–567, 2006.

[53] H.D. Mogal, H. Qian, S.S. Sapatnekar, and K. Bazargan. Clustering based pruning for statistical criticality computation under process variations. In *Proc. Int. Conf. on Computer Aided Design*, pages 340–343. IEEE, 2007.

[54] H.D. Mogal, H. Qian, S.S. Sapatnekar, and K. Bazargan. Fast and accurate statistical criticality computation under process variations. *IEEE*

Trans. on Computer-Aided Design of Integrated Circuits and Systems, 28(3):350–363, 2009.

[55] S Natarajan, A. Krishnamachary, E Chiprout, and R Galivanche. Path coverage based functional test generation for processor marginality validation. In *Proc. Int. Test Conf.*, pages 1–9, 2010.

[56] P. Nigh and A. Gattiker. Test method evaluation experiments & data. In *Proc. Int. Test Conf.*, pages 454–463, 2000.

[57] M. Orshansky and K. Keutzer. A general probabilistic framework for worst case timing analysis. In *Proc. Design Automation Conf.*, pages 556–561, 2002.

[58] A. Ramalingam, G.J. Nam, A.K. Singh, M. Orshansky, S.R. Nassif, and D.Z. Pan. An accurate sparse matrix based framework for statistical static timing analysis. In *Proc. Int. Conf. on Computer Aided Design*, pages 231–236, 2006.

[59] M. W. Robert and P. K. Lala. Algorithm to detect reconvergent fanout in logic circuits. *IEE Proc. Part E Computers and Digital Techniques*, pages 105–111, 1987.

[60] L. Scheffer. Explicit computation of performance as a function of process variation. In *IEEE/ACM Int. Workshop on on Timing Issues in the Specification and Synthesis of Digital Systems (TAU)*, pages 1–8, 2002.

[61] L. Scheffer. Why are timing estimates so uncertain? what could we do about this? In *IEEE/ACM Int. Workshop on on Timing Issues in the Specification and Synthesis of Digital Systems (TAU)*, 2002.

[62] M. Sharma and J.H. Patel. Bounding circuit delay by testing a very small subset of paths. In *Proc. VLSI Test Sympo.*, pages 333–341, 2000.

[63] M. Sharma and J.H. Patel. Finding a small set of longest testable paths that cover every gate. In *Proc. Int. Test Conf.*, pages 974–982, 2002.

[64] M. Sharma and J.H. Patel. What does robust testing a subset of paths, tell us about the untested paths in the circuit? In *Proc. VLSI Test Sympo.*, pages 31–36, 2004.

[65] D. Sinha and H. Zhou. A unified framework for statistical timing analysis with coupling and multiple input switching. In *Proc. Int. Conf. on Computer Aided Design*, pages 837–843, 2005.

[66] D. Sinha and H. Zhou. Statistical timing analysis with coupling. *Proc. Int. Conf. on Computer Aided Design*, 25(12):2965–2975, 2006.

[67] D. Sinha, H. Zhou, and N.V. Shenoy. Advances in computation of the maximum of a set of random variables. In *Proceedings of the 7th International Symposium on Quality Electronic Design*, pages 306–311. IEEE Computer Society, 2006.

[68] D. Sinha, H. Zhou, and N.V. Shenoy. Advances in computation of the maximum of a set of Gaussian random variables. *IEEE Trans. on*

Computer-Aided Design of Integrated Circuits and Systems, 26(8):1522–1533, 2007.

[69] S. Tani, M. Teramoto, T. Fukazawa, and K. Matsuhiro. Efficient path selection for delay testing based on partial path evaluation. In *Proc. VLSI Test Sympo.*, pages 188–193, 1998.

[70] S. Tsukiyama, M. Tanaka, and M. Fukui. A statistical static timing analysis considering correlations between delays. In *Proceedings of the 2001 Asia and South Pacific Design Automation Conference*, page 358. ACM, 2001.

[71] C. Visweswariah, K. Ravindran, K. Kalafala, S.G. Walker, S. Narayan, D.K. Beece, J. Piaget, N. Venkateswaran, and J.G. Hemmett. First-order incremental block-based statistical timing analysis. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, (10):2170–2180, 2006.

[72] L.C. Wang, J. J. Liou, and K.T. Cheng. Critical path selection for delay fault testing based upon a statistical timing model. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 23(11):1550–1565, 2004.

[73] J. Xiong, Y. Shi, V. Zolotov, and C. Visweswariah. Pre-ATPG path selection for near optimal post-ATPG process space coverage. In *Proc. Int. Conf. on Computer Aided Design*, pages 89–96, 2009.

[74] J. Xiong, Y. Shi, V. Zolotov, and C. Visweswariah. Statistical multilayer process space coverage for at-speed test. In *Proc. Design Automation Conf.*, pages 340–345, 2009.

[75] J. Xiong, C. Visweswariah, and V. Zolotov. Statistical ordering of correlated timing quantities and its application for path ranking. In *Proceedings of the 46th Annual Design Automation Conference*, pages 122–125. ACM, 2009.

[76] J. Xiong, V. Zolotov, N. Venkateswaran, and C. Visweswariah. Criticality computation in parameterized statistical timing. In *Proc. Design Automation Conf.*, pages 63–68, 2006.

[77] J. Xiong, V. Zolotov, and C. Visweswariah. Incremental criticality and yield gradients. In *Proc. Design, Automation and Test in Eurpoe*, pages 1130–1135, 2008.

[78] J. Xiong, V. Zolotov, C. Visweswariah, and P.A. Habitz. Optimal margin computation for at-speed test. In *Proc. Design, Automation and Test in Eurpoe*, pages 622–627, 2008.

[79] J. Xiong, V. Zolotov, C. Visweswariah, and N. Venkateswaran. Criticality computation in parameterized statistical timing. In *Proc. Design Automation Conf.*, pages 63–68, 2006.

[80] Shiy Xu and E. Edi. A new way of detecting reconvergent fanout branch pairs in logic circuits. In *Proc. Asian Test Sympo.*, pages 354–357, 2004.

[81] J.S. Yang and N.A. Touba. Automated Selection of Signals to Observe for Efficient Silicon Debug. In *Proc. of VLSI Test Symposium*, pages 79–84.

[82] Y. Zhan, A.J. Strojwas, X. Li, L.T. Pileggi, D. Newmark, and M. Sharma. Correlation-aware statistical timing analysis with non-gaussian delay distributions. In *Proceedings of the 42nd annual Design Automation Conference*, page 82. ACM, 2005.

[83] Y. Zhan, AJ Strojwas, M. Sharma, and D. Newmark. Statistical critical path analysis considering correlations. In *Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design*, page 704. IEEE Computer Society, 2005.

[84] L. Zhang, W. Chen, Y. Hu, and C. Chen. Statistical timing analysis with extended pseudo-canonical timing model. In *Proc. Design, Automation and Test in Eurpoe*, pages 952–957, 2005.

[85] L. Zhang, W. Chen, Y. Hu, and C.C. Chen. Statistical static timing analysis with conditional linear MAX/MIN approximation and extended canonical timing model. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 25(6):1183–1191, 2006.

[86] L. Zhang, W. Chen, Y. Hu, J.A. Gubner, and C.C.P. Chen. Correlation-preserved non-gaussian statistical timing analysis with quadratic timing model. In *Proceedings of the 42nd annual Design Automation Conference*, page 88. ACM, 2005.

[87] L. Zhang, Y. Hu, and C. Chen. Block based statistical timing analysis with extended canonical timing model. In *Proc. Asia and South Pacific Design Automation Conf.*, pages 250–253, 2005.

[88] L. Zhang, Y. Hu, and C. Chen. Statistical timing analysis with path reconvergence and spatial correlations. In *Proc. Design, Automation and Test in Eurpoe*, pages 528–532, 2006.

[89] V. Zolotov, J. Xiong, H. Fatemi, and C. Visweswariah. Statistical path selection for at-speed test. In *Proc. Int. Conf. on Computer Aided Design*, pages 624–631, 2008.

Vita

Jaeyong Chung was born in Seoul, Republic of Korea in 1981. He received the Bachelor of Science degree from Yonsei University in 2006, where his major was Electrical Engineering and his minor was Computer Science. He joined the University of Texas at Austin in 2006, where he received the Master of Science degree in Electrical and Computer Engineering in 2008 and continued to pursue his Ph.D. degree. During the program at UT, he worked as a summer intern at Strategic CAD Lab (SCL), Intel and IBM T.J. Watson Research Center in 2008 and 2010, respectively. His research interests include statistical static timing analysis, robust design, and VLSI testing. The impacts of his research were recognized with Best Paper Nominations in ICCAD 2009 and ASPDAC 2010.

Permanent address: jwings@gmail.com

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.